

01 | 2012

KOS.content

Ergebnisse der Untersuchungen des
Kompetenzzentrum Open Source der DHBW-Stuttgart

Sommer 2012
band.1

INHALT BAND.1

Inhalt __

Beständigkeit eines Open Source Projekts - Analyse von Anerkennungssystemen in Open Source Projekten __ 1

Development of a Model Evaluating the Maturity of Open Source Software __ 103

Entwicklung eines Modells zur Bewertung von Open Source Produktenhinsichtlich des Produktiven Einsatzes __ 183

Leitfaden zur Einführung von Open Source Software in Organisationen __ 245

Das Kompetenzzentrum Open Source (KOS)

Ziel des Projektes

Das Projekt Kompetenzzentrum Open Source der DHBW Stuttgart wurde mit der Zielsetzung ins Leben gerufen, die Einsatzfelder für Open Source Software in Unternehmen zu identifizieren und durch den Einsatz quelloffener Produkte und deren kostengünstigen Einsatzmöglichkeiten Optimierungen in ausgewählten Geschäftsbereichen zu erzielen.

Dies bedeutet konkret, dass z.B. Open Source Software evaluiert wird, um Lizenzkosten zu reduzieren, bewertet wird, ob sie diverse Qualitätskriterien erfüllt und erfolgreich(er) und effizient(er) in Unternehmen genutzt werden kann. Das Ziel des Projektes ist es hierbei, allgemeingültige Lösungskonzepte für Problemstellungen zu erarbeiten, welche von den am Projekt beteiligten Unternehmen zu firmenspezifischen Lösungen weiterentwickelt werden können. Die beteiligten Unternehmen partizipieren so an den Ergebnissen des Projekts.

Zusammenarbeit mit den Dualen Partnern

Die Zusammenarbeit mit den Dualen Partnern gestaltet sich entlang deren Anforderungen und Bedürfnissen. Sie sind die Themengeber für betriebliche Fragestellungen, die im Rahmen des Projekts untersucht werden. Die DHBW steuert die wissenschaftliche, didaktische und methodische Expertise und Forschungskompetenz bei und untersucht die identifizierten Themenfelder.









Im Rahmen des Projektes steuert die DHBW Stuttgart die wissenschaftliche Expertise und Forschungskompetenz bei zur Bearbeitung der betrieblichen Fragestellungen der Dualen Partner. Es entstehen generische Lösungen, welche von den Partnern an Ihre Situation angepasst werden kann.
Im Rahmen der Arbeit entstehen (generische) Lösungen, an denen die Partner teilhaben können indem sie diese auf ihre spezifische Unternehmenssituation anpassen. Zudem fließen die Ergebnisse in die Arbeit der DHBW ein, sodass hier dem Anspruch an eine hohe Anwendungs- und Transferorientierung ganz im Sinne einer kooperativen Forschung Rechnung getragen wird.

An den Ergebnissen des Projekts partizipieren die Dualen Partner Allianz Deutschland AG, die Deutsche Rentenversicherung Baden-Württemberg und die HALLESCHE Krankenversicherung a.G.

Beständigkeit eines Open Source Projektes

-

Analyse von Anerkennungssystemen in Open Source Projekten

16.01.2012

von

Robert Bruchhardt, Anne Golembowska, Maximilian Heinemeyer,
Felix Kugler, Stephen Said, Jennifer Zohar

Duale Hochschule Baden-Württemberg Stuttgart

Inhalt

1 Einleitung ... 1
  1.1 Vorwort ... 1
  1.2 Zielsetzung ... 2
  1.3 Anerkennung als Voraussetzung für Motivation ... 3
2 Grundlagen ... 7
  2.1 Open Source ... 7
  2.2 Open Source Projekt ... 8
  2.3 Open Source Community ... 8
  2.4 Beständigkeit ... 10
3 Methodisches Vorgehen ... 12
  3.1 Evaluationsmethodik ... 12
  3.2 Abgrenzung der Stichprobe ... 12
  3.3 Auswahl der Stichprobe ... 18
4 Auswertung der Stichprobe ... 20
  4.1 Qualitative Analyse ... 20
  4.2 Quantitative Analyse ... 22
  4.3 Kombination von Anerkennungskriterien ... 28
  4.4 Ableitung von Anerkennungsmaßnahmen ... 30
  4.5 Entwicklung eines Werkzeugs ... 35
    4.5.1 Organisatorischen Unterstützung eigenständiger Projekte ... 36
    4.5.2 Finanzielle Kompensation an Entwickler ... 37
    4.5.3 Persönliche Vorstellung ... 38
    4.5.4 Aufstiegsmöglichkeiten ... 39
    4.5.5 Auszeichnungen ... 40
  4.6 Netzdarstellung ... 42
  4.7 Praktische Anwendung ... 44
5 Grenzen der Arbeit ... 49
6 Fazit ... 51
Anhang ... 52
Quellenverzeichnis ... 95

Abbildungsverzeichnis

Abbildung 1 – Anerkennung als Motivationsfaktor 4
Abbildung 2 - Open Source Cycle 9
Abbildung 3 – Verteilung der Anerkennungskriterien in der Voranalyse 22
Abbildung 4 - Finanzielle Kompensation 24
Abbildung 5 - Entwicklungslaufbahn 24
Abbildung 6 - Messen und Events 25
Abbildung 7 - Namentliche Würdigung 25
Abbildung 8 - Individuelle Würdigung 26
Abbildung 9 – Subcommunities 27
Abbildung 10 - Netzdarstellung Auswertung 42
Abbildung 11 – Netzdarstellung Beispielprojekt 43
Abbildung 12 – Netzdarstellung jQuery 45
Abbildung 13 – Netzdarstellung GNU Hurd 46
Abbildung 14 – Verbreitung der Anerkennungsmaßnahmen des Anerkennungsrasters 47

Tabellenverzeichnis

Tabelle 1 Vorabkriterien 13
Tabelle 2 Stichprobenübersicht 19
Tabelle 3 Analyse von Kriterienkombinationen 28
Tabelle 4 Einteilung des Grads der Ausprägung des Anerkennungssystems 35
Tabelle 5 Anerkennungsraster jQuery 45
Tabelle 6 Anerkennungsraster GNU Hurd 46

1 Einleitung

1.1 Vorwort

Die immer weiter steigende Popularität, Professionalisierung und Qualität von Open Source Projekten lässt Open Source Software (OSS) mehr und mehr in den Fokus von Unternehmen als Alternative zu proprietären Lösungen geraten.
Nicht nur große Projekte wie Apache, welches mittlerweile bereits von mehr als jedem fünften Fortune 500 Unternehmen eingesetzt wird, finden wachsenden Zuspruch bei Unternehmen jeglicher Größe. In einem weiteren Rahmen außerhalb der Fortune 500 Unternehmen ist sogar jeder zweite Web Server ein Apache. Auch andere Projekte wie MySQL, FireFox, Linux und die Android Plattform erfreuten sich in unmittelbarer Vergangenheit eines Aufschwungs und steigender Beliebtheit, sowohl im privaten als auch im professionellen Umfeld.
Auf den ersten Blick sind den Entscheidungsträgern in vielen Organisationen die Vorteile von OSS offensichtlich: Keine Kosten durch Lizenzgebühren, öffentliche Dokumentation, die Möglichkeit die Software auf die eigenen Bedürfnisse anzupassen und eventuell sogar weitergehend zu vermarkten.
Bei einer zweiten Betrachtung wird jedoch klar, dass im OSS Umfeld viele Fallstricke und Herausforderungen bestehen können: Komplexe Lizenzrichtlinien, die beachtet und eingehalten werden müssen, eine hohe Dynamik im Release- und Entwicklungsumfeld, eine für traditionelle Firmen ungewohnte Entwicklergemeinschaft und Organisationsstrukturen, sowie nicht zuletzt die Unvorhersehbarkeit der Lebenszyklen von OSS.
Doch auch ohne diese Aspekte vollständig abschätzen zu können, stellt sich für einen Entscheidungsträger immer wieder die Frage:

> *Können wir die OSS Lösung implementieren und auch mittel- bis langfristig Unterstützung aus der Entwicklergemeinschaft - der Community - erwarten, oder stehen wir in einem halben Jahr nach der Migration plötzlich ohne Support da?*

Die Frage der Beständigkeit von OSS Communities ist nicht nur für strategische Unternehmensfragen oder Analysen interessant, sondern kann auch für weitere Forschung eine Rolle spielen. Gibt es Faktoren, Aspekte oder Werkzeuge und Maßnahmen, welche den langfristigen Bestand einer solchen Community fördern und stärken? Kann man anhand solcher oder ähnlicher Kriterien abschätzen, wie beständig zukünftig ein Open Source Projekt sein wird?

1.2 Zielsetzung

Die vorliegende Arbeit liefert einen Beitrag zur Klärung der Frage, wie die Beständigkeit von Open Source Projekten bewertet werden kann. Beständigkeit im Sinne von langfristiger Existenz bzw. langfristigem Erfolg von OSS-Projekten wird in der Literatur maßgeblich über die Motivation der Mitarbeiter erklärt. Ein Teilaspekt von Motivation ist die Anerkennung, die einer Person zuteilwird. Die Aussprache von Anerkennung kann direkt (z.B. Lob) oder indirekt (z.B. Erweiterung der Befugnisse, Honorar) erfolgen. Aus methodischer Sicht tragen u.a. die nachfolgend gelisteten Säulen eines OSS-Projekts zur Anerkennungsförderung bei. Diese ergeben sich aus den Schlussfolgerungen verschiedener aktueller Untersuchungen von OSS-Projekten. [1] [2] [3]

- Soziales System
- Führungssystem
- Entwicklungsmethodik und technische Unterstützung
- Finanzielle Mittel

Das Ziel dieser Arbeit ist es daher zu klären, wie eine Ausgestaltung dieser Säulen aussehen könnte und dafür einen konkreten Kriterienkatalog aufzubauen. Als Grundlage dieses Kriterienkatalogs erfolgt eine breitgefächerte Analyse ausgewählter OSS-Projekte aus verschiedenen Software-Kategorien.

[1] Vgl. Schweik, C. M. et al (2009), S.11

[2] Vgl. Fang, Y./ Neufeld, D (2009), S. 13f

[3] Vgl. Frendo T. (2009), S. 6

Die Adressaten dieser Arbeit sind sowohl Wissenschaftler, welche die Ergebnisse dieser Arbeit als Grundlage weiterer Forschung verwenden können, als auch Entscheidungsträger verschiedenster Art in Unternehmen. Die aus dieser Arbeit resultierenden Entscheidungshilfen können sowohl Management-Entscheidungen beeinflussen als auch Einfluss auf das operative Tagesgeschäft haben.

1.3 Anerkennung als Voraussetzung für Motivation

Zur Erklärung von nachhaltiger Existenz und nachhaltigem Erfolg eines OSS-Projekts existieren eine Vielzahl unterschiedlicher Ansätze und somit auch Ergebnisse. Einer dieser Ansätze ist die Erklärung des Zusammenhangs zwischen der Aussprache von Anerkennung und Motivation. Die Rückführung des Erfolgs eines OSS-Projekts auf die Motivation seiner Mitarbeiter ist eine Schnittmenge verschiedener Arbeiten, die sich mit ersterem Thema auseinander gesetzt haben. Insofern ist es ebenfalls von Interesse, die Einflussgrößen auf Motivation im Bereich OSS zu erarbeiten. Einer der Einflussfaktoren auf die Motivation eines Menschen eine bestimmte Tätigkeit auszuführen, ist die Anerkennung, die ihm durch diese zu Teil wird. Gerade wenn die Tätigkeit keine Aussicht auf direkte Belohnung verspricht. Die Erfahrung von Anerkennung kann intrinsisch als auch extrinsisch erfolgen. D.h. ein Mensch kann sich und seine Taten vor sich selber anerkennen und wertschätzen, aber dies auch durch andere Menschen, d.h. seine externe Umgebung, erfahren. Deshalb ist für die Führung bzw. die Gemeinschaft eines OSS-Projekts wichtig diesen Sachverhalt zu erkennen und in konkreten Maßnahmen zu operationalisieren.

Ferner ist die Stimulation von Motivation durch weiche Faktoren, wie z.B. Anerkennung, für ein OSS-Projekt besonders wichtig, da keine harte juristische Grundlage (z.B. ein vertraglich geregeltes Arbeitsverhältnis) existiert, welches eine Person an die Mitarbeit in einem Projekt bzw. in einem Betrieb bindet. Die Mitarbeit ist an die stets selbstbestimmte innerliche Bindung eines Menschen an eine Idee gekoppelt.

Der Zusammenhang von Anerkennung und langfristiger Existenz bzw. Erfolg eines OSS-Projekts ist in nachfolgender Abbildung 1 dargestellt. Die Abbildung beschreibt das System von Einflussgrüßen, welches die Grundlage der in dieser Arbeit beschriebenen Untersuchung bildet. Wie zu sehen ist, existiert zwischen langfristiger Motivation und nachhaltiger, d.h. langfristiger

Mitgliederbeteiligung ein Zusammenhang.[4] Letzterer wiederum kann als einer der primären Erfolgsfaktoren für die Nachhaltigkeit eines OSS-Projekts gesehen werden. Generell jedoch, sind es nicht die Faktoren langfristiger Motivation, sondern die der kurzen, die eine Person zur Mitarbeit bewegen. Nun ist die Frage wie die Beweggründe sich verschieben bzw. verschieben lassen können. Bei der Betrachtung der langfristigen Motivationsgründe fällt auf, dass diese extrinsisch gefördert werden können. Psychologische Bedürfnisse wie Anerkennung oder Selbstverwirklichung können z.B. durch die Umgebung einer Person beeinflusst werden.

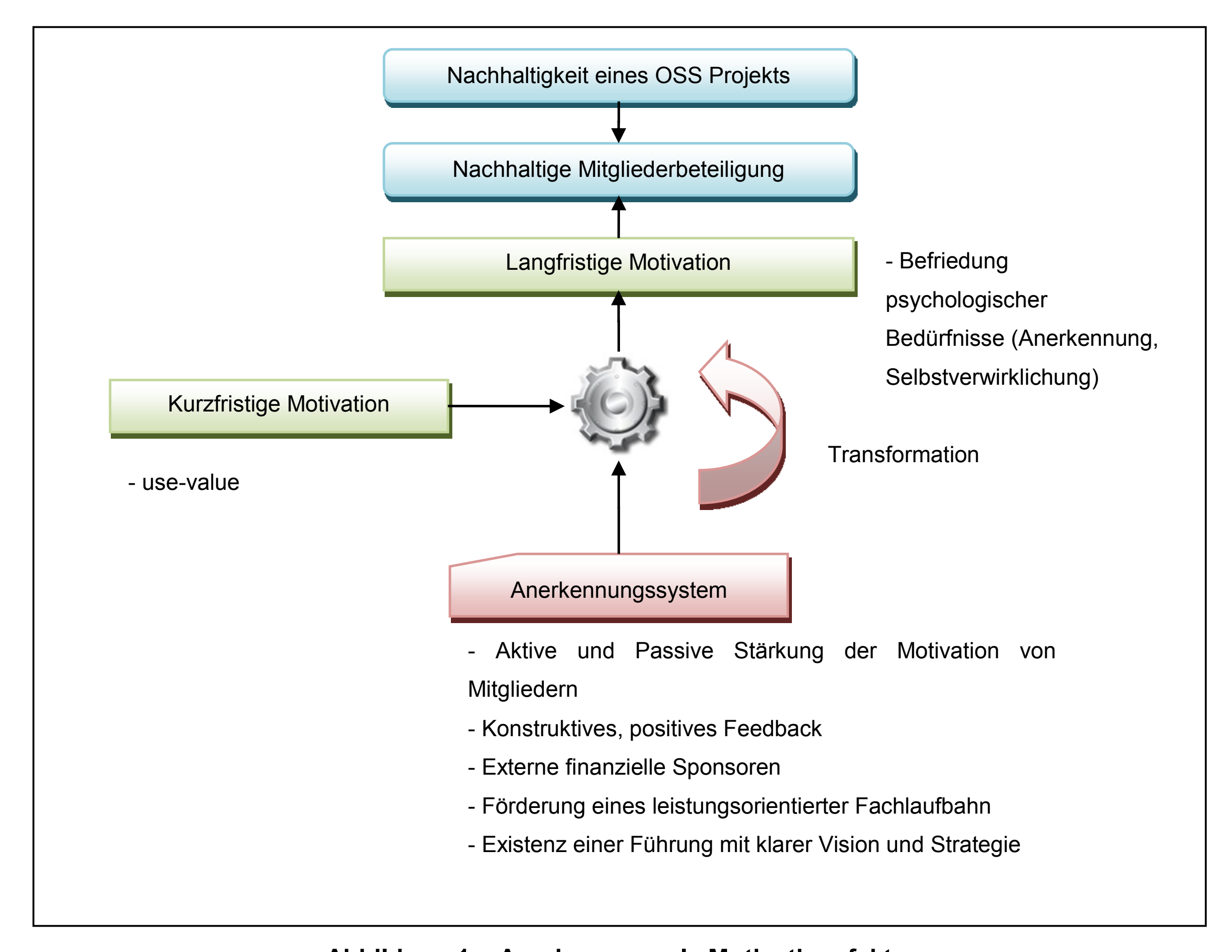


Abbildung 1 – Anerkennung als Motivationsfaktor

Die grundsätzliche Annahme ist, dass die Elemente Führungssystem, soziales System und Vergütung eines OSS-Projekts die Motivation zur Mitarbeit über die Aussprache von Anerkennung beeinflussen können. D.h. dass operationale Wege existieren, Anerkennung auszudrücken. Die Rolle des Führungssystems eines OSS-Projekts wird durch Yulin Fang und

[4] Vgl. Fang, Y./ Neufeld, D (2009), S. 13ff

Derrick Neufeld in ihrer Arbeit über nachhaltige Mitarbeit in OSS Projekten aufgegriffen.[5] In einer Analyse des phpMyAdmin Projekts sehen sie dieses in der Verantwortung die Umgebung für die langfristige Teilnahme eines Projekts zu schaffen. Charles M. Schweik und weitere Forscher der University of Massachusetts fokussieren sich auf die Aspekte ‚Leadership, Vision, Utility, Community'. Sie sehen das Führungssystem eines Projektes ebenfalls als maßgeblichen Erfolgsfaktor an.[6] Dieses muss, wie auch in einer kommerziellen Unternehmung, eine klare Strategie haben und repräsentieren. Steven Weber weist in seiner Arbeit auf verschiedene Führungsstile im Open Source Bereich hin. Jeder dieser Stile wirkt sich anders auf die für Erfolg des Projektes maßgebende Faktoren aus. Ein solcher Faktor ist die Motivation der Entwickler.

Diese Meinung teilen Fang und Neufeld, die feststellen, dass die Führung eine der wesentlichen Säulen ist, die zu langfristigem Projekterfolg beiträgt. Weiter stellen diese fest, dass das soziale System des Projekts, d.h. die Organisation der Gemeinschaft der Entwickler, dazu beitragen muss, stetig Motivation zu stimulieren. Rogers und Shoemaker zeigen in ihrer Ausarbeitung, dass es eine enge Verknüpfung von Sozialem System und Anerkennung gibt. Wenn Mitglieder in eine Gemeinschaft reinwachsen entsteht eine Bindung zu den übrigen Mitgliedern der Gemeinschaft. Das einzelne Mitglied wünscht sich Anerkennung und Status in dem Sozialen System und wird so zum Engagement motiviert.

Dies kann nach Fang u.a. dadurch geschehen, dass eine im Sinne der Paradigmen der Legitimate peripheral participation Theorie (LPP) günstige Lernatmosphäre herrscht. Diese beschreibt wie neue Mitglieder einer Arbeitsgemeinschaft sich entwickeln und zu tragenden Säulen dieser werden. Dieser Punkt ist wichtig, da Fang und Neufeld weiter argumentieren, dass ein Erfolgsfaktor eines OSS Projekts ist, dass die Beweggründe der Mitarbeit von Faktoren kurzfristiger Motivation in die von langfristiger überführt werden können. Diese seien grundsätzlich verschieden. Nach dem Ergebnis einer Studie werden 80% aller Projekte auf Grund von **fehlender Langzeitmotivation** abgebrochen.[7] In der kurzen Sicht sei es primär das Bedürfnis nach der Lösung eines technischen Problems (use-value), das Menschen zur Mitarbeit an einem OSS-Projekt bewegt. Langfristig jedoch, ist es primär die **Befriedigung** psychologischer Bedürfnisse wie **Anerkennung**, Selbstverwirklichung oder Individualität (joy, interaction).

Zur Stimulation dieser Art von Motivation können gemäß Fang und Neufeld verschiedene Säulen beitragen. Dazu zählt z.B. die Führung sowie Organisation eines OSS-Projekts.

---

[5] Vgl. Fang, Y./ Neufeld, D (2009), S.8

[6] Vgl. Schweik, C. M. et al (2009), S. 12

[7] Vgl. Colazo, J. / Fang, Y. (2009)

Abschließend müssen ebenfalls finanzielle Aspekte, wie z.B. Honorarzahlungen berücksichtig werden. Diesen Punkt sehen auch Roberts et al. in ihrer Langzeituntersuchung des Apache Projekts als ausschlaggebend an. Sie empfehlen jedem OSS-Projekt finanzielle Mittel anzunehmen und unter Anderem an ihre Mitglieder auszuschütten. Dies sei förderlich für Stimulation von Motivationsfaktoren, die nach Fang und Neufeld als Faktoren langfristiger Motivation angesehen werden können.

2 Grundlagen

2.1 Open Source

Als Open Source wird eine Entwicklungsmethode beschrieben, die auf zwei Säulen aufbaut. Zum einen wird das Prinzip der Transparenz umgesetzt[8]. Das heißt, dass der Code für jeden Nutzer für das Kopieren und bearbeiten frei zugänglich sein muss. Daraus folgt der Vorteil, dass zum Beispiel Fachleute oder Experten die Möglichkeit haben, die Arbeit zu überprüfen und zu begutachten. Für die effektive Nutzung und Bearbeitung bilden sich häufig Communities, die den Code weiterentwickeln und verbessern. Zusammenfassend lassen sich drei wesentlich Punkte festlegen, die Open Source definieren: der frei verfügbare Code, die entstehende Community und die Möglichkeit der freien Kopie und Bearbeitung des Codes.

Ergänzend gibt es eine Definition der Open Source Initiative (OSI), die auf bestimmten Merkmalen aufbauen, welche auf der Website definiert werden[9]. Dabei gibt es zehn Punkte, die erfüllt werden müssen. Zum Beispiel muss die Software unter einer von der Open Source Initiative bestätigte Lizenz veröffentlicht werden[10]. Das beinhaltet die Idee der freien Verteilung der Quellen und des Binärcodes. Dabei sind das Kopieren und Modifizieren am Code erlaubt und sogar erwünscht. Das bedeutet die Nutzer haben das Recht die Software zu nutzen, kopieren, verteilen, studieren, ändern und verbessern. Die OSI verlangt des Weiteren, dass die Integrität des Quellcodes des Autors bestehen bleibt und keine Person oder Gruppe und auch kein Geschäftsfeld diskriminiert wird. Die Lizenz darf auch nicht auf ein Produkt spezialisiert sein und auch keine anderen Softwareprodukte in ihren Rechten beschneiden. Darüber hinaus muss die Lizenz technologisch neutral sein und verteilt werden

Durch dieses Vorgehen lassen sich verschiedene Vorteile finden. Zuerst einmal wird durch das Transparenzprinzip eine bessere Code Qualität erreicht wird, da Fehler und Ungenauigkeiten eher entdeckt werden können. Durch die bessere Qualität des Codes erhoffen sich die Anwender eine erhöhte Ausfallsicherheit. Mittels technologischer Neutralität wird mehr Flexibilität zu geringeren Kosten erzielt. Gleichzeitig versprechen sich die Anwender eine höhere Unabhängigkeit von bestimmten Anbietern.

[8] Präsentation: Open Source Software Research

[9] Vgl. Radcliffe, M. (o.Ja), http://opensource.org/docs/osd

[10] Vgl. Radcliffe, M. (o.Jb) http://opensource.org/licenses/alphabetical

2.2 Open Source Projekt

Ein Projekt ist im Allgemeinen laut DIN 69901 "...ein Vorhaben, das im Wesentlichen durch die Einmaligkeit der Bedingungen in ihrer Gesamtheit gekennzeichnet ist...". Dazu zählen bestimmte Bedingungen, etwa finanziell oder durch knappe Ressourcen, als auch ein bestimmtes Ziel (DIN 69901). Ein Open Source Projekt ist daher ein Vorhaben mit dem Ziel, eine spezifische, quelloffene Software zu erstellen. Dabei sind die limitierenden Faktoren vor allem die finanziellen und humanen Ressourcen, da es durch die freie Vertreibung des Produktes meist schwierig ist, an finanzielle Mittel zu gelangen.

Die Abgrenzung zu anderen Projekten generell ist dabei insbesondere, dass ein Open Source Projekt nicht zeitlich abgeschlossen ist, d.h., dass ständig an weiteren Überarbeitungen oder zusätzliche Funktionalitäten gearbeitet wird. Auch hängen die verfügbaren Ressourcen im großen Maße von der Beteiligung der Community ab.

In der folgenden Arbeit wird der Begriff eines Open Source Projektes als Umfang aller Bestandteile und Beteiligten einer bestimmten Open Source Software verstanden. Dazu zählen insbesondere die Community nebst eingesetzten Kommunikationslösungen, das Programm selbst und die Benutzer.

2.3 Open Source Community

Allgemein gibt es viele Definitionen für den Begriff „Community". Unter Entwicklungscommunity wird der Zusammenschluss von Entwicklern, die kollaborativ mit gleichen Zielen und Motiven an einer Software-Entwicklung arbeiten, verstanden. Die Software-Entwicklung kann anschließend „mit freiem Zugang zum Quellcode liberal lizenziert, kostenlos genutzt und auch verändert werden"[11]. Die Schwierigkeit liegt im Aufbau bzw. in der Aufrechterhaltung einer Community. Ohne eine Community können Software Projekte wie z.B. Open Source Projekte nicht zum Erfolg geführt bzw. überhaupt realisiert werden. In diesem Projekt wird die Entwicklungscommunity als Zusammenschluss von Entwicklern im Open Source Bereich (Open Source Community) verstanden.

Grundsätzlich werden zwei Arten von Open Source Communities unterschieden. Zum einen gibt es Firmen, welche ihre Mitarbeiter dafür bezahlen in Open Source Communities aktiv

[11] Vgl. S. Schaffert/ D. Wieden-Bischof (2009), S.120

mitzuwirken mit dem Ziel das Open Source Projekte voran zu treiben. Zum anderen werden viele Communities von Entwicklern gepflegt, die unentgeltlich und freiwillig an einem Projekt arbeiten.[12]

Die folgende Abbildung, Abbildung 2, stellt die Arbeitsweise einer Community an einem Open Source Projekt in einem Kreislauf dar.

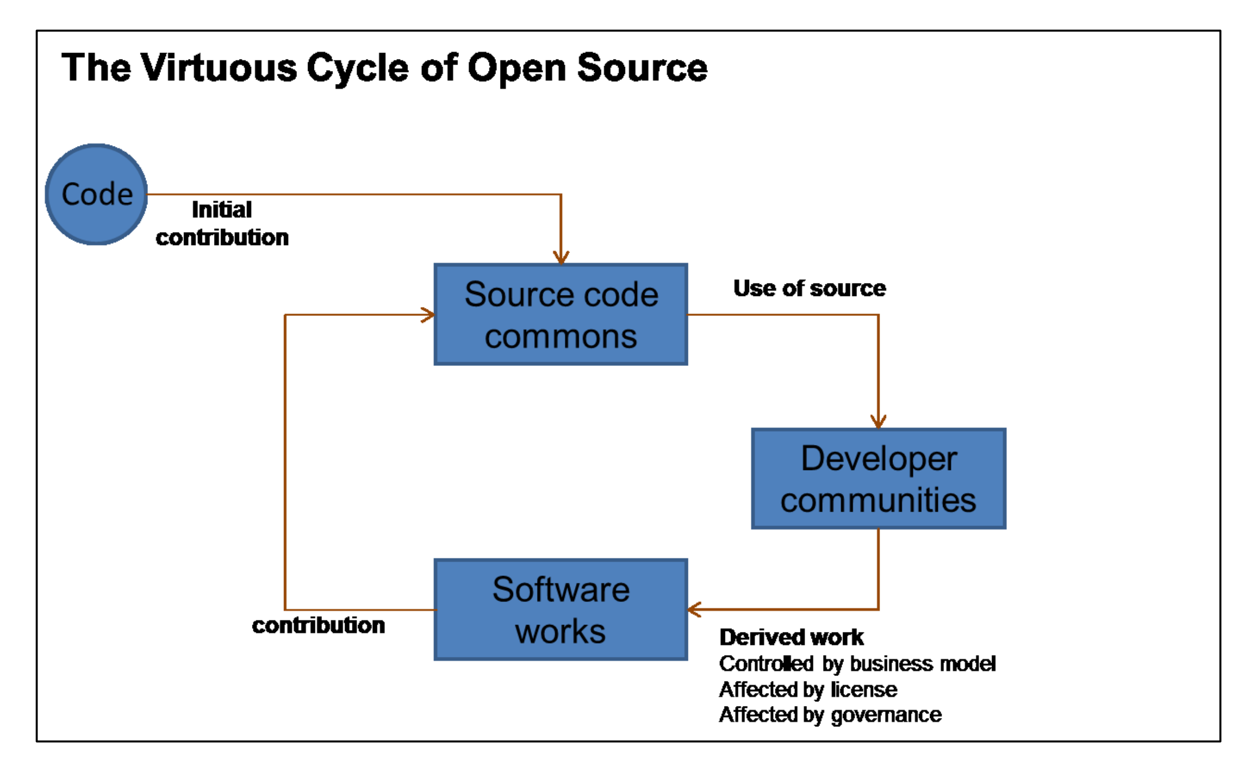


Abbildung 2 - Open Source Cycle

Wie Abbildung 2 zeigt, folgen Open Source Communities einer bestimmten Dynamik. Die Arbeit an einem Projekt beginnt mit einem Entwickler, der auf Grund von bestimmten Motiven den Quellcode eines Programmes veröffentlicht. Die Software wird als ein Grundgerüst (Source Code Commons) online frei für andere zur Verfügung gestellt, um darauf aufbauend eine Community zu errichten.

Nachdem ein allgemeiner Startpunkt geschaffen wurde haben auch andere Entwickler die Chance an der Software zu arbeiten und Fehler zu beheben (Developer Communities). Dabei werden die unterschiedlichen Arbeitspakete unter den einzelnen Entwicklern oder Entwicklerteams in der Community verteilt. (derived work) Geschäftsmodelle, welche unter anderem ein Strategie-, Resourcen- und Netzwerkmodell enthalten, tragen zu einer funktionierenden Software bei (controlled by business models). Die Lizenz beeinflusst die Arbeit

[12] Vgl. H. Klapf/ R. Pölsch (2010)

an der Software zusätzlich. Beispielsweise gibt es Open Source Projekte welche sowohl unter kostenlosen als auch kostenpflichtigen Lizenzen vertrieben werden. Bei kostenpflichtigen Lizenzen wirken meist bezahlte Entwickler an dem Projekt mit, wodurch ein gewisser Fortbestand des Projektes garantiert werden kann (affected by license). Zusätzlich affektiert die Regierung allgemein das Internet, z.B. im Bereich Datenschutz, was den Erfolg eines Softwareprojekts mitbestimmt (affected by governance). Allerdings endet der Prozess nicht an dieser Stelle, da auf Grund von Codeänderungen, Testläufe und Pflege des gesamten Codes (Software Works) erforderlich sind. Der von Entwicklern modifizierte Code bildet nun das neue Grundgerüst, welches wiederum von der Community weiterentwickelt bzw. modifiziert werden kann. [13]

2.4 Beständigkeit

In der Literatur existieren diverse Definitionen für den Begriff Beständigkeit in Bezug auf Open Source Software. Es muss an dieser Stelle jedoch klar abgegrenzt werden, welches Begriffsverständnis im Kontext dieser Arbeit anzuwenden ist.

Einerseits kann der Begriff als „Informationelle Nachhaltigkeit“ verstanden werden, der von der 1987 entwickelten, ursprünglichen Definition des Wortes „Nachhaltigkeit“ ausgeht. Veröffentlicht im Brundtland-Bericht unter dem Titel „Our Common Future“ bezeichnet das Konzept *eine „Entwicklung, die die Bedürfnisse der Gegenwart befriedigt, ohne zu riskieren, daß künftige Generationen ihre eigenen Bedürfnisse nicht befriedigen können.“*

Obwohl die Erläuterung dieser Definition den Fokus vor allem auf Wirtschaftlichkeit, Ökologie und Gesellschaft legt, kann daraus für den Bezug zu Open Source und informationeller Nachhaltigkeit das Kriterium der Übertragbarkeit abgeleitet werden (OSS Jahrbuch 2008, S. 113): Entwicklungen sollen nicht nur heutige, sondern auch zukünftige Bedürfnisse befriedigen können. Im Vergleich zu kommerzieller Software sei laut Grassmuck freie Software geeigneter, die Übertragbarkeit zu gewährleisten, da Beschränkungen (z.B: Copyright, Patente, und kommerzielle Interessen der Stakeholder) weitestgehend eliminiert werden. Dadurch, dass ein kommerzieller Vertrieb von Open Source Software immer noch möglich und rentabel ist, wie IT-Konzerne wie Google, IBM oder HP zeigen, bedarf es nicht ausschließlich gemeinnützig

[13] Vgl. Oracle Corporation (2010), http://java.sun.com/developer/technicalArticles/javase/opensource_phipps

agierender Personen, um einen Kapitalstock von Wissen und Quellcode aufzubauen, der weltweit für jedermann zugänglich ist. Dieser weltweite Zugang erschließt in Entwicklungsländern neue Möglichkeiten in Hinblick auf Bildung, Gesellschaft und Wirtschaft und erfüllt somit die interpretierte Definition von Beständigkeit.

Trotz des Bewusstseins über die Wichtigkeit des Aspekts soll im Kontext dieser Arbeit Beständigkeit weniger als Verpflichtung der Gesellschaft gegenüber angesehen werden. Der Fokus wird stattdessen im ersten Schritt auf die Essenz bzw. Grundlage der genannten Faktoren, demzufolge ausschließlich auf den „**Fortbestand**“ des betreffenden Open Source Projektes für zukünftige Generationen gelegt. Weitere Faktoren wie Wirtschaftlichkeit ergeben sich nach dem Verständnis der Autoren dieses Artikels automatisch aus diesem Kriterium. Aus diesem Grund sollte die sich auf einen Gesichtspunkt von Beständigkeit beschränkende Auslegung nicht als Simplifizierung, sondern als Priorisierung verstanden werden: Ist ein Fortbestand eines Projektes nicht gewährleistet, lohnt sich keine Investition und kein Einsatz der Software und damit wäre eine Betrachtung aller anderen Faktoren überflüssig.

3 Methodisches Vorgehen

3.1 Evaluationsmethodik

Die in dieser Arbeit dokumentierte Untersuchung ist in drei Schritte gegliedert:

1) *Vorbereitung der Studie*
Auf Basis der Ergebnisse bestehender Forschung wird eine Auswahl von Eigenschaften eines OSS-Projekts erstellt, die dazu dient die Stichprobe der Studie abzugrenzen und für eine Untersuchung interessante OSS-Projekte zu ermitteln. Modelle der Verhaltens- als auch Lerntheorie dienen dazu die in der Analyse betrachteten Anerkennungsmethoden eines Projekts zu bestimmen.

2) *Analyse*
Die ausgewählten Projekte werden betrachtet und die Ergebnisse der Untersuchung dokumentiert.

3) *Entwicklung eines Anerkennungssystems*
Die Ergebnisse der zuvor durchgeführten Analyse werden auf Gemeinsamkeiten hin analysiert und konsolidiert. Innerhalb der Ergebnisse wird versucht Zusammenhänge aufzuzeigen und zu erklären. Das entwickelte gemeinsame Verständnis von Anerkennungsaussprache wird in einem System zusammengetragen und versucht über einen Kriterienkatalog konkret prüfbar zu machen.

3.2 Abgrenzung der Stichprobe

Um eine tiefergehende Analyse von Motivationsaspekten nicht an beliebigen Open Source Projekten anzuwenden, muss zunächst eine Vorauswahl erfolgen. Dazu werden im Folgenden als wichtig erachtete und schnell verfügbare Kriterien definiert. Diese wurden zum großen Teil aus gängigen Modellen übernommen. Anhand dieser Vorabkriterien können dann Projekte evaluiert werden, die im folgenden Kapitel in Bezug auf Motivation näher beleuchtet werden.

Jedes Vorabkriterium kann dabei als *unzureichend*, *genügend*, und überdurchschnittlich bewertet werden.
Je nach dem Grad wird von einer individuellen Punktzahl ein Punkt abgezogen (unzureichend), nichts verändert (genügend) oder ein Punkt hinzugefügt (überdurchschnittlich). Eine Punktzahl über oder gleich null weist somit auf ein tendenziell erfolgreiches Open Source Programm hin.
In der folgenden Tabelle werden alle Vorabkriterien aufgeführt. Danach werden sie näher erläutert.

Kriterium	unzureichend (-1)	Genügend (0)	Überdurchschnittlich (+1)
Alter der Community	<1	1 - 2	>2
Google Blog Search (nur bei eindeutigem Namen)	<50.000	50.000-200.000	>200.000
Existenz im t3n Top 100 http://t3n.de/opensource/top100/	-	Nein	Ja
Dokumentation	Nein	Ja	-
Anzahl Downloads	<1000/Woche	1000-50.000/Woche	>50.000/Woche
Userrating	<60%	60%-85%	>85%
Anzahl der Entwickler	<3	4 - 7	>7
Sponsoren vorhanden	-	Nein	Ja
Anzahl der verwendeten Sprachen	>2	1 - 2	-
Anzahl Forenbeiträge all time	<5000	5000-100.000	>100.000

Tabelle 1 Vorabkriterien

Alter der Community

Das Alter der Community ist ein wesentlicher Faktor im Bereich der Pre-Selektion. Eine junge Community ist im Bezug auf die Nachhaltigkeit tendenziell wesentlich risikoreicher einzuschätzen als ein schon 3 Jahre altes Projekt. Diese Aussage kann aus mehreren Gründen getroffen werden:

- Tendenziell liegt bei älteren Projekten eine höhere Bekanntheit und damit eine größere Anzahl an aktiven Benutzern vor
- Langes Fortbestehen weist auf eine aktive Projektleitung oder zumindest auf eine effektive Organisation hin
- Wahrscheinlich haben sich über einen längeren Zeitraum schon Strukturen und Methoden gebildet, die näher analysiert werden können
- Junge Projekte sind in vielen Fällen weder in der Methodik, noch in der Zielstruktur oder im Bereich der eigentlichen Software ausgereift
-

Auch das QSOS-Modell verwendet dieses Kriterium zur Evaluierung eines Open Source Projekts. [14]

Google Blog Search (nur bei eindeutigem Namen)

Die Anzahl an Treffern bei einer Suche mit Google Blogs gibt die Popularität in Fachkreisen in einem gewissen Maße wieder. Je größer die Trefferanzahl bei der Suche nach einem bestimmten Open Source Projekt Namen, desto größer das Interesse von Blogschreibern und Fachkreisen an der Software und der Community. Das liefert zum einen Rückschlüsse auf die Größe der Community und Benutzer, als auch Schlüsse in Bezug auf Relevanz des Projektes.
Dieses Kriterium macht jedoch nur Sinn, wenn der Name des Projektes eindeutig ist. Es muss ausgeschlossen werden, dass bei einer Suche nach dem Namen nur darauf bezogene Artikel mitgezählt werden.
Auch der interne Algorithmus für die OSS-Top 100 des Open Source Fachmagazins t3n wertet dieses Kriterium aus. [15]

Existenz im t3n Top 100

Um zusätzlich zu anderen Kriterien herauszufinden, ob das zu untersuchende Projekt ein in Fachkreisen bekanntes und gut bewertetes ist, wird zusätzlich evaluiert, ob es in den Top 100 des Open Source Fachmagazins t3n gelistet wird. Wenn es nicht gelistet ist, wird kein Punkt vom spezifischen Rating abgezogen, denn hier werden natürlich nur subjektive Top-Projekte

[14] Vgl. o.V. (2006), http://www.qsos.org/download/qsos-1.6-en.pdf, S 16

[15] Vgl. yeebase media GmbH (2005), http://t3n.de/opensource/top100/

gelistet. Wird das entsprechende Projekt jedoch gelistet, wirkt sich dies positiv auf die Gesamtpunktzahl aus, denn es ist davon auszugehen, dass ein hier genanntes Projekt von Fachleuten für gut befunden wurde. Außerdem ist es wahrscheinlich aktuell und beliebt.

Dokumentation

Das Vorhandensein einer Dokumentation ist ein wesentlicher Faktor für ein funktionierendes Open Source Projekt und weist wesentlich auf die Kommunikation in der Community des entsprechenden Projektes hin. Wenn das Vorgehen innerhalb des zu evaluierenden Projektes nicht schriftlich festgehalten wird, wird es neuen Benutzern erschwert, der Community beizutreten. Auch die Benutzung und Weiterentwicklung der Software wird erheblich erschwert. Somit wirkt es sich negativ auf die Vorabbewertung aus, wenn keine Dokumentation vorhanden ist. Die Existenz einer solchen dagegen wird im Rahmen dieser Arbeit als selbstverständlich angenommen und gibt keinen Pluspunkt.
Das QSOS-Modell verwendet auch dieses Kriterium zur Evaluierung eines Open Source Projekts im Bereich “Industrialised solution”. [16]

Anzahl Downloads

Anhand der Quantität an Downloads, hier überprüft pro Woche, kann abgeleitet werden, wie viele Benutzer Interesse an der Software des Projektes haben, auch wenn diese vielleicht nicht unmittelbar zum Projekt beitragen. Jedoch sagt das Interesse über ein Open Source Projekt aus, für wie viele Anwender die Software zumindest in Frage kam. Je populärer das Programm dabei ist, desto höher ist auch die Wahrscheinlichkeit, dass aktive Entwickler und Interessanten an diesem Projekt beteiligt sind, da eine breite Masse an Benutzer erreicht werden kann. Insbesondere kann über dieses Kriterium die “stille Benutzermenge” gemessen werden, also die Anwender, die das Programm gebrauchen, sich jedoch nicht aktiv angemeldet oder innerhalb des Projektes kommuniziert haben.

---

[16] Vgl. o.V. (2006), http://www.qsos.org/download/qsos-1.6-en.pdf, S 19

Userrating

Große Open Source Portale wie Source Forge bieten Benutzern eine schnelle und einfache Möglichkeit der Bewertung eines bestimmten Open Source Programmes an. [17] In dieser Arbeit wird, falls vorhanden, diese Bewertung als weiteres Vorselektierungskriterium angewendet, um Rückschlüsse auf die fachliche Qualität des Programmes ziehen zu können. Ein gut bewertetes Programm funktioniert wahrscheinlich auch gut und weitgehend fehlerfrei, sodass in gewissem Maße davon ausgegangen werden kann, dass eine kompetente und effektive Community an diesem Projekt beteiligt ist. Ist die Bewertung schlecht, kann über die Mangelhaftigkeit der Software aus Endbenutzersicht auch auf die des Projektes in irgendeiner Form geschlossen werden.

Anzahl der Entwickler

Dieses Kriterium wurde aus dem QSOS-Modell entnommen und wertet aus, wie viele aktive Entwickler an einem Projekt beteiligt sind.[18] Folgerichtig gilt hierbei, dass je größer die Anzahl der Entwickler, desto geringer ist die Wahrscheinlichkeit, dass das Projekt von wenigen Mitgliedern getragen wird und somit nicht an deren Partizipation gebunden ist. Auch weisen viele Entwickler auf einen großen Pool an Expertise, Kapazitäten und gesteigertem Interesse hin. Eine geringe Anzahl an Entwicklern erhöht also das Risiko eines Totalausfalls durch Absprung eines oder mehrerer Kern-Entwickler und weist auf tendenziell geringe Kapazitäten hin, welches beide wesentliche Faktoren in Bezug auf Nachhaltigkeit und Potenzial eines Open Source Projektes sind.

Sponsoren vorhanden

Das Vorhandensein von Sponsoren innerhalb eines Open Source Projektes ist im ersten Schritt ein Indikator für ein gesteigertes Interesse auch außerhalb der Entwicklergemeinde. Es kann davon ausgegangen werden, dass Sponsoren entweder eigene Entwickler für das Projekt bereitstellen, oder aber (auch ohne direkte Bereitstellung eines Marketingbudgets) durch eigene

[17] Vgl. Source forge (o.J.), http://sourceforge.net/

[18] Vgl. o.V. (2006), http://www.qsos.org/download/qsos-1.6-en.pdf, S 18

Bekanntheit auch die Bekanntheit des Projektes steigern. Engagement von Sponsoren bedeutet gleichzeitig auch, dass diese eine langfristige Perspektive in der Entwicklung sehen, was exemplarisch an der Unterstützung des Joomla! Projekts durch diverse Firmen deutlich gemacht werden kann.[19]
Es bleibt jedoch zu bemerken, dass Sponsoren sich normalerweise erst ab einer gewissen Projektgröße bzw. einer gewissen Projektlebenszeit engagieren. Aus diesem Grund soll das Nichtvorhandensein von Sponsoren nicht als negativ, umgekehrt jedoch das Vorhandensein als positiv beurteilt werden.

Anzahl der verwendeten Sprachen

Die Anzahl der bei der Entwicklung verwendeten Programmiersprachen kann als Zeichen für die Konsistenz eines OS Projektes verstanden werden. Darüber hinaus - und nach Meinung der Autoren als wichtiger zu bewerten - sorgt die einheitliche Verwendung einer einzigen bzw. einer geringen Zahl von Sprachen dafür, dass potentielle Entwickler für die Mitarbeit am Projekt ausschließlich Kenntnisse in dieser einen Sprache besitzen müssen. Die Menge dieser Entwickler ist statistisch betrachtet höher als diejenige, die aus Entwicklern mit erhöhten / diversifizierteren Kenntnissen besteht. Dem Projekt stehen so potentiell mehr Entwickler zur Verfügung, wobei eine erhöhte Entwicklerzahl als positiv für die Beständigkeit der Community angesehen wird [20].

Gesamtanzahl von Foreneinträgen

Die Anzahl aller Forenbeiträge innerhalb eines Communityforums lässt aus mehreren Gründen Aussagen über den Erfolg eines Projekts zu:
Erstens zeugt eine hohe Anzahl Beiträge generell von einem langen Bestand der Community. Zweitens kann davon ausgegangen werden, dass, sofern das Projekt noch aktiv ist, bei einer insgesamt hohen Anzahl an Beiträgen auch eine rege Aktivität im Forum herrscht und diese das Projekt unterstützt (zumindest eine Stagnation verhindert). Drittens lässt die Anzahl der Beiträge

---

[19] Vgl. Joomla (o.J.), http://www.joomla.org/about-the-joomla-project/sponsorships/development-sponsors.html

auch indirekt auf die Diversität der abgehandelten Themen schließen, was im Rahmen dieser Arbeit aufgrund einer möglichen Beteiligung eines größeren Personenkreises als ebenfalls positiv angesehen wird.
Es darf jedoch bei einer Betrachtung des Kriteriums nicht vergessen werden, die allgemeine Projektlebenszeit in Relation zur Anzahl der Beiträge zu setzen. Das Nichtvorhandensein eines Forums bzw. eines anderen Diskussionsmediums wird aufgrund der Simplizität und als notwendig angesehener Implementierung für ein Projekt als negativ bewertet.

3.3 Auswahl der Stichprobe

Zur Analyse werden wie vorhergehend erwähnt Open Source Projekte aus verschiedenen Kategorien ausgewählt. Um eine möglichst breite Abdeckung von Typen zu gewährleisten, orientiert sich diese Arbeit an anerkannten Aufstellungen wie die Open Source Projekt Auflistung des Online Magazins t3n[21], die Suchfunktion des Repository-Portals sourceforge [22] und die Sammlung nach Downloadanzahl des Online-Magazins Chip.de.[23]
Ausgehend von dieser und weiterer individuellen Recherche wurden erfolgreiche, populäre und durch ihre Prozesse oder Community besondere Open Source Projekte zu 5 Hauptkategorien akquiriert. Diese Kategorien sind insbesondere unter der Berücksichtigung der **Relevanz für Unternehmen** gewählt. Dazu zählt diese Arbeit Softwareprojekte zur standardisierten Erstellung von Web-Inhalten, diverse Office-Lösungen, Betriebssystemalternativen, Verwaltungsprogramme und Werkzeuge für die Zusammenarbeit. Diese Kategorien wichtig für Unternehmen sein, denn sie bilden eine günstige Alternative für teure Lizenz-Software. Eine Evaluation seitens Unternehmen in diesen Kategorien um bestehende Software auszutauschen ist somit wahrscheinlich und die Analyse von Anerkennungsprozessen in diesen naheliegend, um die üblichen, gängigen Wertschätzungsmodelle ausfindig zu machen.

[21] Vgl. o.V. (o.J.), http://t3n.de/opensource/projects/

[22] Vgl. Source forge (o.J.), http://sourceforge.net/

[23] Vgl. ChipOnline (o.J.), http://www.chip.de/bildergalerie/Top-50-Open-Source-Downloads-September-2011-Galerie_47813697.html

Kategorie	Kurzbeschreibung	Gewählte Stichprobe
Kollaborations-Systeme	Unter Kollaborations-Systemen wird Software verstanden, die die Zusammenarbeit zwischen Menschen erleichtert, die über verschiedene Systeme verteilt sind. Dies wird erreicht durch Kommunikationsplattformen (Chat, Forum), Email-Clients, Instant Messaging oder Wikis.	Thunderbird Pidgin Openfire Kunagi Foswiki
Betriebssysteme	Betriebssysteme stellen die Grundfunktionen eines Computersystems sicher. Sie allokieren die Rechnerressourcen und kontrollieren die Programmausführungen. Im Bereich Open Source stechen hier vor allem die Linux-Systeme hervor.	Fedora Ubuntu ArchLinux GNU Hurd JNode
Content Management Systeme	Ein Content Management System ist der englischen Entsprechung nach ein Inhaltsverwaltungsystem. Es wird verwendet, um im Verbund mit anderen diverse Inhalte, z.B. einfachen Text oder komplexere Dateien, zu verwalten und bearbeiten. Sie werden hauptsächlich zur Webseiten-Erstellung verwendet und setzen keine Kenntnisse einer bestimmten Programmiersprache wie php voraus.	Typo3 Joomla Drupal Xoops Jahia
Administrations-Systeme	Administrationssysteme bezeichnen allgemein Programme und Systeme, die zur Verwaltung von Datenmengen genutzt werden. Dabei wurdem im Rahmen dieser Arbeit Anwendungs- und Webserver, ein relationales Datenbankverwaltungssystem, ein Privatsphärenschutztool, virtuelle Maschinenverwaltung und ein Business Intelligence Tool analysiert.	JBoss Application Server Apache HTTP Server mySQL Palo Suite BleachBit Xen
Office & Multimedia	Unter Office & Multimedia werden im Rahmen dieser Arbeit Programme verstanden, die zur Bearbeitung, Sichtung und Verwaltung von Dateien und Medien genutzt werden. Dazu wurden Open-Source Projekte im Bereich der Büroarbeit, der Dateikomprimierung, des Webbrowsers und der Graphenerstellung überprüft.	LibreOffice 7zip Firefox Tulip K-Meleon
Entwicklung	Unter der Kategorie Entwicklung werden in dieser Arbeit alle Open-Source-Systeme analysiert, die bei der Softwareentwicklung unterstützen. Dazu zählen erweiterte Text- und Codebearbeitungsprogramme, Entwicklungs-Frameworks und Compiler.	Notepad++ Eclipse Rails FOXOpen EiffelStudio

Tabelle 2 Stichprobenübersicht

4 Auswertung der Stichprobe

4.1 Qualitative Analyse

Vorab ist zu erwähnen, dass alle Stichproben mit einer kurzen Einleitung und einem Fazit in Anhang 1 gefunden werden können.

Die Betrachtung der erhobenen Daten zeigt, dass z.B im Bereich von Kollaborationssoftware durch das Projekt Openfire zu erkennen ist, dass die Erfahrung von Anerkennung durch außerordentliche persönliche Würdigung eines Entwicklers in der Öffentlichkeit, ein generell auffälliger Aspekt ist. Innerhalb von Openfire werden einzelne Personen durch Auszeichnungen wie „**Developer of the month**" aus der Masse herausgehoben und für ihre Mitarbeit am Projekt gelobt. Damit werden viele der Entwickler in der Motivation an einem OSS-Projekt mitzuarbeiten in besonderem Maße gestärkt, da ihre Motivation oftmals keinen finanziellen Hintergrund hat. Vielmehr basiert sie auf der Selbstverwirklichung innerhalb einer Gemeinschaft, die sich nicht an kapitalistischen Idealen orientiert, sondern dem Menschen die Möglichkeit bieten möchte seine Interessen in großer Selbstbestimmung umzusetzen. Dies setzt z.B. auch Eclipse sehr gut um, welches regelmäßig über „Feature of the month" ein Subprojekt besonders hervorhebt und damit die Leistungen der zugehörigen Entwickler betont. Zudem existiert hier eine Art von „**Hall of Fame**" in der inaktive oder ausgeschiedenen Personen für ihre Verdienste „gerühmt" werden. Auch aus dem Bereich Middleware kann beispielsweise JBoss genannt werden, welches „Top Performer" herausstellt.
Weiter ist zu sehen, dass viele Projekte, wie z.B. Fedora Linux, es nicht nur geschickt wissen Lob auszusprechen, sondern durch das Herausheben einer Person aus der Anonymität einer Entwicklercommunity, auch den menschlichen Konkurrenztrieb fördern. Egoismus und Konkurrenzkampf sind Teile des menschlichen Wesens und treiben diesen dazu an, bestimmten Tätigkeiten nachzugehen. Fedora veranstaltet auf Messen und Kongressen **Entwicklerwettbewerbe**. Ähnliches wird auch im Ubuntu Projekt durchgeführt, in welchem Designwettbewerbe dazu führen, dass Grafiker zu überdurchschnittlichen Leistungen angespornt werden. Auch das sehr erfolgreiche Eclipse Projekt setzt Anerkennungsmaßnahmen dieser Art um, so existierte wohl für eine gewisse Zeit eine Serie aus monatlichen Programmier- und Designwettbewerben. Aus Sicht der Führung eines Projekts sind Maßnahmen dieser Art eine geschickte Möglichkeit zum einen die Qualität des entwickelten Produkts als auch die Motivation der Mitarbeiter zu steigern. Durch Konkurrenzkampf werden Menschen oftmals zu

außerordentlich guten Leistungen angetrieben. Daneben bedeutet das Kräftemessen einen Kampf um Prestige und Anerkennung innerhalb der Gemeinschaft von Projektmitgliedern.

In Kombination mit diesen beiden Aspekten von Anerkennungsmaßnahmen ist bei vielen erfolgreichen Projekten eine Art von **Fachlaufbahn** zu entdecken. Die Verantwortlichkeiten und Befugnissen werden hier in einer Hierarchie abgebildet und motivieren dadurch Entwickler ihren Arbeitseinsatz zu steigern und möglicherweise in dieser aufzusteigen. Die bereits erwähnten Projekte jQuery und Ruby on Rails schaffen es z.B. Mitglieder der höchsten Hierarchiestufen zu heroisieren und glorifizieren. Im Projekt Ruby on Rails existiert ein sogenanntes „Core Team“, welches sich öffentlich selbst als „made men of Ruby on Rails“ präsentiert. Innerhalb von jQuery tragen die Mitglieder des Boards Titel wie „jQuery Core Lead“ oder „Standards Lead“. Für Menschen sind Bezeichnung dieser Art wichtig, da sie damit Anerkennung, Reputation und letztlich Macht verbinden. Ferner ist auch hinsichtlich des Aspekts der Fachlaufbahn zu sagen, dass der menschliche Wettbewerbstrieb mit der verbundenen Aussicht auf Anerkennung zu erkennen ist. Seine Signifikanz zeigt sich über verschiedene OSS-Projekte und Themenbereiche hinweg. In unsere Studie ist, wie bereits gesagt zu erkennen, dass sich dieser in Anerkennungsmaßnahmen wie Titeln oder öffentlichen Lob in einer Vielzahl von bekannten und erfolgreichen Projekten äußert.

Ein weiteres erkennbarer Muster ist die Existenz von **persönlichen Treffen** wie Messen, Kongressen oder Stammtische. Das Projekt Drupal organisiert z.B. Großveranstaltungen an denen neben den Mitgliedern der Community, Sponsoren des Projekts teilnehmen. Dadurch erhalten diese Veranstaltungen einen hohen Stellenwert, welcher die Teilnahme zu etwas besonderem werden lässt. Die Teilnehmer, sprich Entwickler oder sonstige Projetmitglieder, fühlen besondere Anerkennung dadurch, dass sie an diesen Veranstaltungen teilnehmen dürfen. Auch andere Communities wie z.B. Joomla („Joomladays“), Eclipse („Eclipsecon“) oder Ubuntu („Developer summit“) haben Großveranstaltungen. Ein gängiger Teil dieser Veranstaltungen ist zudem, dass die Teilnehmer ihre Arbeit am OSS-Projekt innerhalb eines Vertrags öffentlich vorstellen. Durch Auftritte vor versammeltem Community Publikum kann sich ein Mensch in seinem Stellenwert innerhalb dieser Community gestärkt fühlen und dadurch Anerkennung erfahren.

Im Folgenden soll die Stichprobe auch quantitativ bewertet werden.

4.2 Quantitative Analyse

Vor einer Hauptanalyse ist es sinnvoll, die Vorabbewertung der einzelnen Projekte näher zu betrachten. Hieraus lassen sich gegebenenfalls Zusammenhänge ableiten, die erkennen lassen, ob bestimmte Analysekriterien nur in einer gewissen Klasse Projekte auftreten.
Die statistische Übersicht der Vorabkriterien zeigt, dass über alle Kriterien hinweg ein breites Spektrum an Werten vertreten ist. Demzufolge kann nicht die Gesamtübersicht betrachtet werden, sondern es müssen Einzelkriterien geprüft werden. Folgende Grafik zeigt die prozentuale Verteilung der Kriterien in der Vorabanalyse:

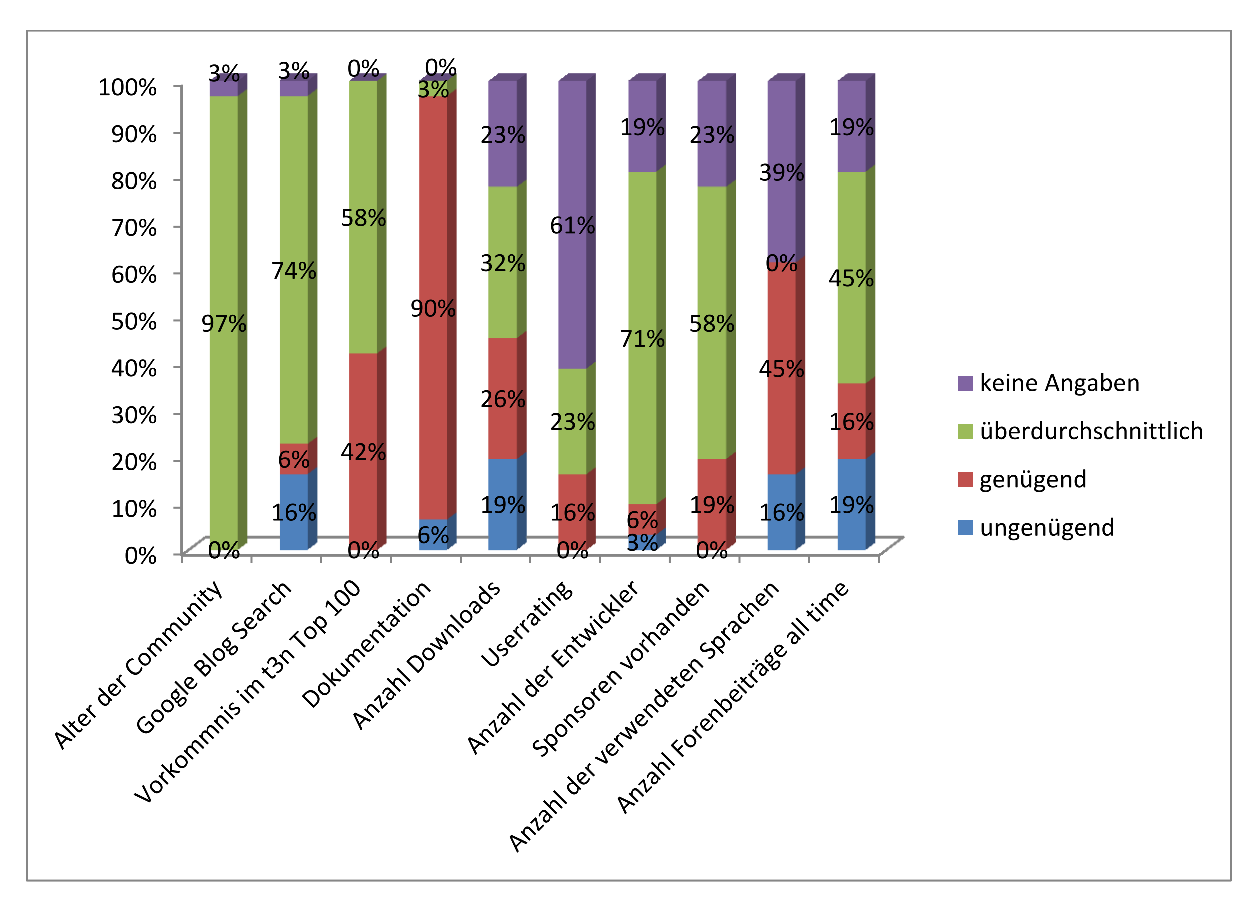


Abbildung 3 – Verteilung der Anerkennungskriterien in der Voranalyse

Es ist zu erkennen, dass in 97% aller Projekte die Community schon einen sehr langen Zeitraum besteht. Dieses Kriterium hat allerdings keine Aussagekraft für Hauptanalysekriterien, da fast ausschließlich „überdurchschnittliche“ Projekte betrachtet wurden. Eine Differenzierung ist hiermit nicht gegeben. Ebenso verhält es sich mit der Dokumentation, die in nahezu allen Projekten vorhanden ist. Dies ist nach Meinung der Autoren dieser Arbeit eine **Grundvoraussetzung**, die für Open Source Software Projekte gegeben sein sollte. Aus diesem Grund ist es wenig sinnvoll, einen Zusammenhang zwischen diesem Kriterium und den Hauptanalysekriterien herstellen zu wollen.

Von erhöhter Relevanz ist eine Betrachtung größerer Projekte. Als solche zu bewerten sind vor allem jene, die die folgenden Kriterien aufweisen:

- 74% aller Projekte, die in den t3n Top 100 vorkommen, weisen auf Grund der externen Bewertung auf eine gewisse Projektgröße hin
- 58% aller Projekte, die durch Sponsoren unterstützt werden, können durch das monetäre Engagement mit hoher Wahrscheinlichkeit als große Projekte bezeichnet werden
- Außerdem weisen 45% aller Projekte eine überdurchschnittliche Anzahl an Forenbeiträgen auf. Diese lassen auf eine breite Mitgliederbasis und eine hohe Aktivität schließen, was wiederum als Indikator für ein großes Projekt betrachtet werden kann.

Festzustellen war in der statistischen Analyse, dass sich kein direkter Zusammenhang zwischen einzelnen Vorabkriterien und der Hauptanalyse herstellen lässt. Gründe hierfür sind vielfältig; als Hauptaspekt zu nennen ist wohl die starke Interdependenz der einzelnen Faktoren zueinander, sodass die Verbindung zweier expliziter Kriterien nicht genügend Aussagekraft besitzt.

Aus diesem Grund werden im Folgenden ausschließlich die Hauptanalysekriterien der betrachteten Projekte statistisch ausgewertet. Hierbei wird der Fokus auf einzelne, wiederkehrende Kategorien gelegt, sodass diese zusammengefasst und näher evaluiert werden können.

An erster Stelle zu nennen ist eine Form der monetären Unterstützung für Entwickler, hier in Abbildung 5, als finanzielle Kompensation bezeichnet.

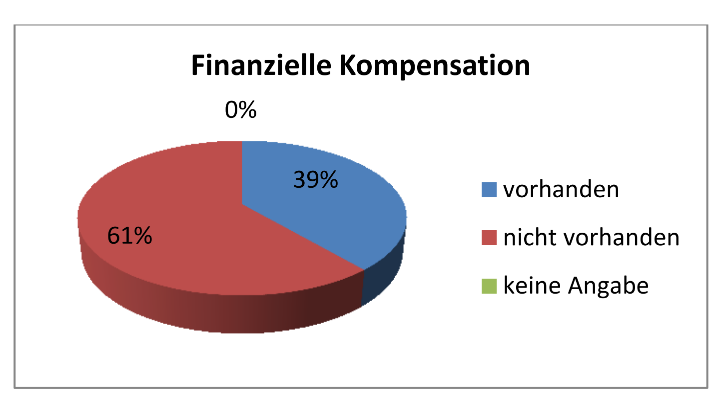


Abbildung 4 - Finanzielle Kompensation

Es wurde deutlich, dass in 39% aller analysierten Projekte eine solche Art der Kompensation vorhanden ist. Hierbei sind unterschiedlichste Ausprägungen vorhanden, die von einer einmaligen **Ausschüttung** eines Betrags durch die Projektführung bis hin zu einer **Festanstellung** mit einem regelmäßigen Gehalt reichen. Obwohl die Mehrheit der Projekte dieses Kriterium nicht aufweist, kann die Aussage erfolgen, dass vor allem in großen Projekten bzw. in Projekten mit Sponsorendeckung dieses Anerkennungssystem vorhanden ist.

Ein weiteres Evaluationskriterium ist die Existenz einer Entwicklungslaufbahn, Abbildung 6.

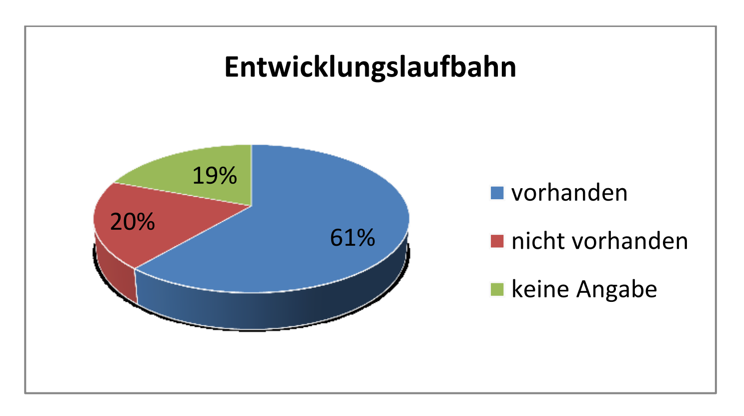


Abbildung 5 - Entwicklungslaufbahn

Auch hier sind unterschiedliche Ausprägungen zu beobachten, die von wenigen, nicht dokumentierten Schritten bis hin zu einem vollständig definierten „Hierarchiesystem“ spannen. In 61% aller betrachteten Projekte ist die Laufbahn, d.h. eine **Karrieremöglichkeit** in einer beliebigen Form gegeben. Es ist also der Schluss zulässig, dass dieses Kriterium von erhöhter Wichtigkeit für die Evaluation des Anerkennungssystems innerhalb einer Community ist.

Die statistische Analyse ergibt desweiteren, dass 65% der betrachteten Projekte auf Messen oder anderen Events präsent sind. Dies wird im Folgenden auf Abbildung 7 dargestellt.

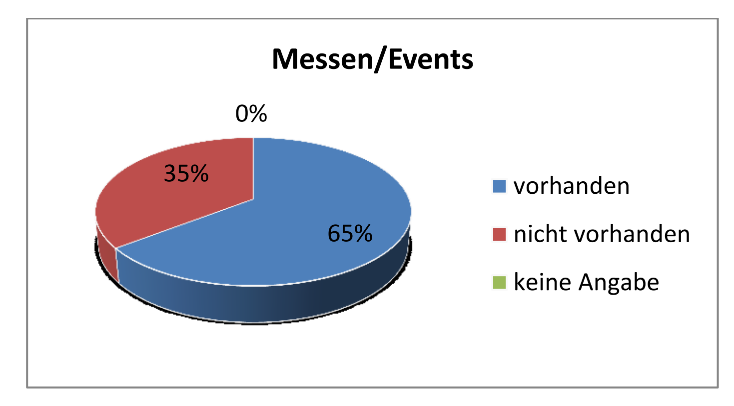


Abbildung 6 - Messen und Events

Hierbei stellt sich jedoch die Frage, inwieweit dieses Kriterium ausschlaggebend für die Anerkennung innerhalb der Gemeinschaft ist. Zwar können sich Mitglieder auf diesen Treffen kennenlernen und sich gegenseitig ihre Projekte vorstellen, allerdings ist dies eher als Motivationskriterium zu sehen: Ist das Projekt auf Messen vertreten, ist ein initiales Engagement wahrscheinlicher. Eine wirkliche Transparenz bzw. erhöhte Anerkennung auf größerer Ebene ergibt sich jedoch erst, wenn eine Community-Mitglied ein Teilprojekt und in diesem Rahmen auch sich selbst präsentiert.

Dieser Aspekt ist im Großen und Ganzen in Abbildung 8 besser zusammengefasst als „Namentliche Würdigung“, wobei in dieser Evaluation auch noch die Nennung der Namen einzelner auf der Website oder im Wiki zu sehen ist.

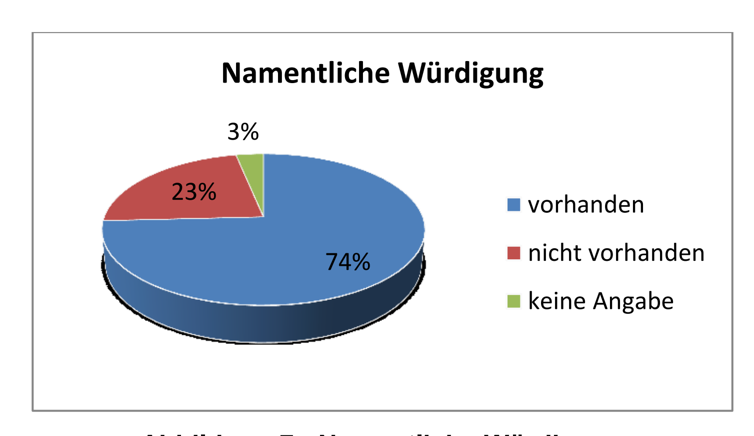


Abbildung 7 - Namentliche Würdigung

Mitglieder können sich hier sowohl intern im Projekt als auch der Allgemeinheit präsentieren und steigern so ihre Bekanntheit, was ihnen bei guten Leistungen Anerkennung verschafft. Unter dieses Kriterium fällt bspw. auch eine **Nennung** eines neuen Mitglieds oder **Vorstellungen** kompletter Teams. Vorteilhaft ist an der namentlichen Würdigung, dass dieses Kriterium nicht notwendigerweise in Zusammenhang mit der Entwicklungslaufbahn gesehen werden muss: Oft ist keine bestimmte Hierarchiestufe notwendig, um die Möglichkeit einer namentlichen Nennung bzw. persönlichen Vorstellung in Anspruch zu nehmen. Insgesamt ist dieser Faktor in 74% aller betrachteten Projekten vorhanden, was für eine deutliche Relevanz auf Anerkennungssysteme insgesamt spricht.

Eine sehr deutliche Form der Anerkennung bietet das nächste Anerkennungskriterium, die Würdigung der Leistung Einzelner (individuelle Auszeichnung), gezeigt in Abbildung 9.

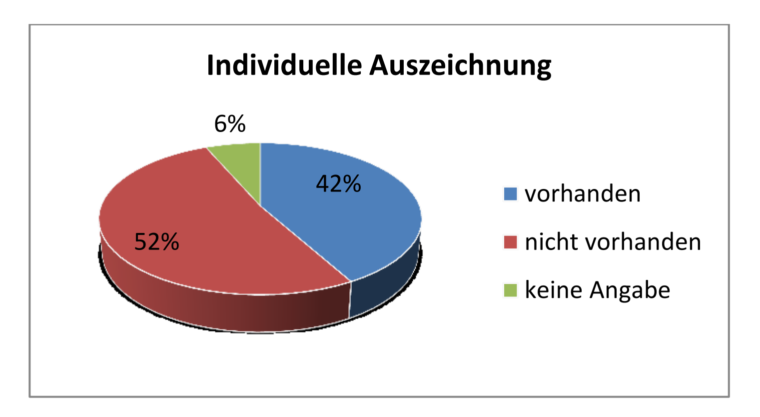


Abbildung 8 - Individuelle Würdigung

Hierbei ist jedoch zu beachten, dass mehrere Evaluationsaspekte der einzelnen Projekte zusammengefasst wurden, um eine Vereinheitlichung und bessere Vergleichbarkeit zu gewährleisten. So wurden die Einzelfaktoren „Feature of the Month", „Developer of the Month" genauso wie die Würdigung einzelner (theoretischer) Subprojekte wie z.B. Paper mit in die Analyse einbezogen. Ein weiteres Kriterium, welches mit in die Analyse einging, ist das Vorhandensein von Wettbewerben jeglicher Art, die mit dem Projekt in Zusammenhang stehen. Hervorzuheben ist dieser Faktor, da er in 29% aller betrachteten Projekte vorzufinden ist. Obwohl nicht durch Zusammenhänge zwischen Vorabkriterien und eigentlichen Analysekriterien belegbar, liegt der Schluss nahe, dass vor allem im Rahmen von sehr großen Projekten Wettbewerbe ausgerichtet werden, wodurch bei einer alleinigen Betrachtung dieser Projekte der Anteil steigen müsste. Insgesamt zeigt die zusammenfassende Analyse, dass über alle Projekte

hinweg 42% die Leistungen einzelner würdigen. Demnach lässt sich auch dieser Punkt als wichtiges Anerkennungskriterium bezeichnen.

Als letzten relevanten Aspekt für die statistische Auswertung hat sich das Vorhandensein von Subprojekten, Abbildung 10, herausgestellt.

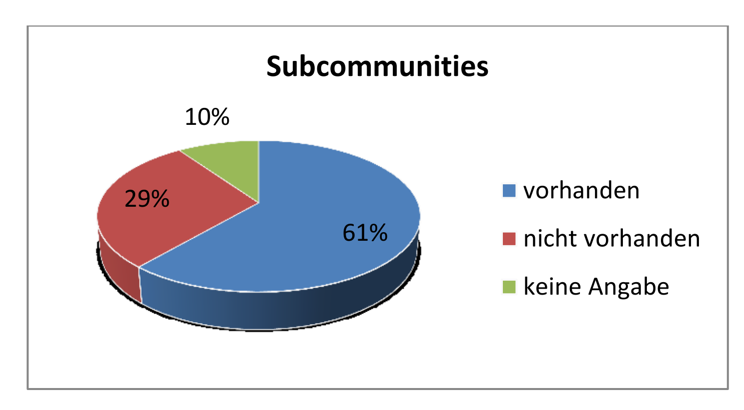


Abbildung 9 – Subcommunities

Hierbei werden sowohl Projekte, die auf das Original aufsetzen, als auch das Vorhandensein von **durch Mitglieder eingerichtete Communities** (z.B. lokale Teams) einbezogen. Es sind unterschiedliche Ausprägungen in der Unterstützung von Subcommunties zu beobachten: Teilweise wird vom Hauptprojekt ausschließlich Platz für ein eigenes Forum bereitgestellt, in anderen Fällen werden sogar komplette Sub-Entwicklungsprojekte bzw. –Gruppen durch finanzielle Mittel unterstützt. Dieser Aspekt, der in 61% aller betrachteten Projekten vorhanden ist, ist als sehr relevant in Bezug auf Anerkennung zu bewerten, da hierdurch nicht nur die Motivation der Mitglieder steigt, sich im Projekt zu engagieren, sondern sie ihre Leistung auch noch darüber hinaus gewürdigt sehen.

4.3 Kombination von Anerkennungskriterien

Nachdem im vorangegangenen Kapitel die wichtigsten bzw. prägnantesten Anerkennungskriterien quantitativ einzeln betrachtet wurden, soll nun statistisch ein **Zusammenhang** zwischen diesen Kriterien aufgezeigt werden. Hierfür werden Kombinationen aus jeweils zwei Kriterien analysiert. Ziel ist es eine konkrete Übersicht über gemeinsam auftretende Anerkennungsmechanismen zu erhalten, um so Unternehmen eine Handlungsempfehlung geben zu können, welche Kriterien bei einer Einführung eines Open Source Projektes zur nachhaltigen Community-Unterstützung implementiert werden sollten. Eine Evaluation einer Verkettung von mehr als zwei Kriterien erscheint im Rahmen dieser Arbeit als wenig sinnvoll: Die Stichprobe von 31 ausgewählten Open Source Projekten wird als zu klein bewertet, um eine konkrete, fundierte Auskunft über eine erhöhte Vorkommnis mehrerer Kriterien geben zu können. Zu Test- und Überprüfungszwecken wurden eine solche Analyse durchgeführt, aufgrund der wenig aussagekräftigen bzw. repräsentativen Ergebnisse allerdings verworfen und nicht in diese Arbeit inkludiert.

Die folgende Tabelle zeigt eine Übersicht über das prozentuale Auftreten zweier Anerkennungskriterien innerhalb der 31 betrachteten Open Source Projekten:

	finanzielle Kompensation	**Entwicklungs-laufbahn**	**Namentliche Würdigung**	**Würdigung einzelner Leistungen**	**Sub-communities**
finanzielle Kompensation	X	32,26%	32,26%	25,81%	35,48%
Entwicklungs-laufbahn	32,26%	X	51,61%	38,71%	48,39%
Namentliche Würdigung	32,26%	51,61%	X	41,94%	48,39%
Würdigung einzelner Leistungen	25,81%	38,71%	41,94%	X	38,71%
Subcommunities	35,48%	48,39%	48,39%	38,71%	X

Tabelle 3 Analyse von Kriterienkombinationen

Die Analyse zeigt auf, welche Kriterien-Kombinationen besonders häufig auftreten. So wird bspw. deutlich, dass in 35,48% aller betrachteten Projekte eine Kombination aus finanzieller Kompensation und der Unterstützung von Subcommunities als wirkungsvolles Anerkennungssystem zu betrachten ist. Ebenso tritt die Kombination aus der Existenz einer Entwicklungslaufbahn und der namentlichen Würdigung einzelner Community-Mitglieder sehr häufig auf (51,61% aller Projekte). Zusätzlich dazu lässt sich beobachten, dass eine Anerkennung spezifischer Nutzerleistungen häufig eingesetzt wird, wenn gleichzeitig auch eine personelle, namentliche Würdigung in der Community erfolgt. Dies ist in 41,94% aller betrachteten Projekte der Fall. Die Aussage wiederum, dass sich hieraus eine Dreierkombination aus Entwicklungslaufbahn, namentlicher Würdigung und Würdigung einzelner Leistungen ergibt, ist statistisch aufgrund der Stichprobenmenge nicht zu belegen und kann an dieser Stelle nur vermutet werden. Die Beschränkung der Untersuchung auf den Zusammenhang zweier Kriterien erscheint aus diesem Grund immer noch sinnvoll. Werden Subcommunities als primär zu untersuchendes Kriterium gewählt, ergibt sich hier vor allem ein Zusammenhang mit der Entwicklungslaufbahn und der namentlichen Würdigung (je 48,39%).

Zusammenfassend stellt die obige Übersichtsmatrix deutlich dar, welche Anerkennungskriterien innerhalb der gewählten Stichprobe verstärkt in Kombination auftreten. Hierbei wurden jedoch keine bestimmten Ausprägungen der jeweiligen Kriterien in Betracht gezogen. Außerdem müssen im weiteren Verlauf dieser Arbeit die einzelnen Anerkennungsfaktoren **individuell**, nicht in Kombination, näher erläutert werden, um vollständig für ein Unternehmen verständlich und demzufolge auch anwendbar zu sein.

Gründe, die das Auftreten bestimmter Kriterien-Kombinationen erläutern, sollen in diesem Rahmen nicht eruiert werden – essentiell ist vor Allem die quantifizierte Feststellung innerhalb der Stichprobe. In Verbindung mit der Erklärung einzelner Kriterien und einer Beschreibung unterschiedlicher Ausprägungsgrade können hieraus dann bereits erste Handlungsempfehlungen für den kombinierten Einsatz bestimmter Anerkennungssysteme in Unternehmen hergeleitet werden, auf die näher in Kapitel 4.6 eingegangen wird.

4.4 Ableitung von Anerkennungsmaßnahmen

Aus der zu vorigen qualitativen und quantitativen Auswertung ergeben sich verschiedene Kriterien, die die Anerkennung der Mitarbeiter maßgeblich beeinflussen. Das folgende Kapitel führt diese aus.

Finanzielle Kompensation

Als eines der wesentlichen Kriterien hat sich die finanzielle Kompensation an die Entwickler gezeigt. In 39% der analysierten Stichproben-Projekte findet dieses Mittel Anwendung. Das ist also knapp bei jedem zweiten Projekt, was bei einer freien Software, die nicht direkt durch jede Nutzung Lizenzgebühren generiert, eine beachtlich hohe Zahl ist. Der monetäre Faktor ist somit immer ein wichtiger Punkt, sowohl im Sinne der Motivation als auch der Anerkennung. Wenn andere Benutzer oder Unternehmen bereit sind, für eine Leistung Geld zu bezahlen, gibt es ihnen offensichtlich einen ganz besonderen Nutzen, der mehr wert ist als der Preis des Produktes. Somit kann der Entwickler direkt feststellen, was seine Leistung "wert" ist, ergo erfährt er Anerkennung. Diese motiviert Entwickler von Projekten langhaltig und weist somit auf etablierte, qualitativ hochwertige und standhafte Projekte hin.

Im Wesentlichen gibt es finanziell drei Ausprägungsformen:

- Jobangebote mit Festgehalt innerhalb des Open Source Projektes
- Die gegebene Möglichkeit für das Projekt kostenpflichtige Erweiterungen zu schreiben (insbesondere in CMS-Projekten vorhanden)
- Ausschüttung von Spenden oder sonstigen Geldern des Projektes direkt an die Entwickler

Wenn ein Open Source Projekt Stellenangebote bereitstellen kann, ist eine besondere Anerkennungsleistung zu erkennen. Ein normaler, bezahlter Mitarbeiter eines Projektes zu sein mit finanzieller Sicherheit gibt wesentlich mehr Wertschätzung und Motivation als unbezahlt und eventuell mit einem anderen Hauptjob nebenbei an dem Projekt zu arbeiten.
Etwas geringer anzusiedeln, aber ebenfalls motivierend und für die Anerkennung einzelner wichtig ist die Möglichkeit, innerhalb eines Open Source Projektes oder darauf aufsetzend auf einer gegebenen Plattformen Erweiterungen zu schreiben und für Geld vertreiben zu können. Bietet ein bestimmtes Projekt eine Plattform dafür oder unterstützt es diesen Prozess in einer Weise, wird den Entwicklern die Möglichkeit gegeben, über einen Preis und ein Bezahlungssystem direkt ihre Anerkennung zu messen.

Eine meist unregelmäßigere Form der monetären Anerkennung ist die der Ausschüttung von Spenden- oder sonstigen Geldern an Aktive der Community. Bekommt man als Entwickler eine unerwartete Geldbelohnung, drückt dies natürlich eine große Wertschätzung seitens des Projektes aus.

Persönliche Vorstellung

Die Analyse der Stichprobe weist auf ein weiteres entscheidendes Kriterium hin: Die der persönlichen Vorstellung. Entwickler und sonstige aktive Mitglieder des Projekts wird die Möglichkeit gegeben, einen kleinen Steckbrief von sich zu veröffentlichen. Dies schafft Anerkennung über die Verbindung der Software und des Projektes mit dem Namen einzelner Aktiver. Dadurch, dass dieses Mittel in 74% der Stichprobe angewandt wurde, wird deutlich, wie oft diese Anerkennungsform innerhalb von Open Source Projekten benutzt wird.
Dieses Kriterium hat dabei je nach Community eine andere Ausprägung. So kann beispielsweise auf der Webseite des Projektes immer jedes neue Mitglied vorgestellt werden. Dies sorgt für eine gewisse Anerkennung gleich zu Beginn des Eintritts eines jeden Entwicklers in die Community und somit für schnelle Initiale Motivation.
Um Mitglieder auch nachhaltig mit Begeisterung an das Projekt zu binden, nutzen Open Source Communities auch Teamvorstellungen und Organisations-Aufstellungen. In diesen wird die Projektstruktur erklärt, etwa in ein Projektleiterteam, ein Designerteam und ein Developer-Team. Darüber hinaus werden dann die einzelnen Mitglieder dieser Gruppen vorgestellt und so genau definiert, wie jeder zu diesem Projekt beiträgt und an welcher Position er steht.
Des Weiteren sorgen persönliche Vorstellungen und Steckbriefe für eine persönlichere Kommunikation zwischen allen Beteiligten, wenn man sich ungefähre Bilder zeichnen kann, wer sich hinter den einzelnen Usernames verbirgt.

Auszeichnung

Eine weitere Möglichkeit der Wertschätzung ist die Auszeichnung von Projektbeteiligten, die durch besondere Leistungen in irgendeiner Form positiv aufgefallen sind. Meist stimmt die Community darüber ab, wer für eine solche Auszeichnung in Frage kommt. Eingesetzt werden Auszeichnungen im Bereich Innovation, Aktivität, Projektzugehörigkeit oder geleisteter Beitrag. Die Geehrten werden dann auf der Seite des Projektes für alle angezeigt und meist vorgestellt. Des Weiteren findet eine “Hall of Fame”, also eine intertemporäre Aufstellung aller Ausgezeichneten, in manchen Projekten Anwendung. Im Rahmen unserer Analyse exisiterten Auszeichnungen für bestimmte Zeiträume, etwa der Entwickler des Monats. Auch hier werden diese in irgendeiner Form hervorgehoben und vorgestellt. Insgesamt wurden in 42% der

Stichprobe Formen von Auszeichnungen gefunden. Also würdigt beinahe jedes zweite Open Source Projekt seine Mitglieder individuell. Dies führt zu dem Schluss, dass dieses Mittel gerne benutzt wird und zur Motivation der Community beitragen kann.
Manchmal findet eine Auszeichnung unter Ausschluss der Öffentlichkeit statt, etwa durch eine private Mail im Namen der Community an ein bestimmtes Mitglied. Durch den Verzicht der öffentlichen Darstellung bietet dies eine potenziell geringere Wertschätzung.
Das Mittel der Auszeichnung ist ein klarer Anerkennungsfaktor. Wenn Aktive für ihre Leistung gewürdigt werden und dies öffentlich zur Schau gestellt wird, motiviert das ungemein. Es wird der breiten Masse an Anwendern und eigener Community mitgeteilt, wer maßgeblich positiv zu der Software oder dem Projekt allgemein beigetragen hat. Die Motivation der Entwickler wird somit gestärkt durch den Fakt, dass eine öffentliche Darstellung durch einen Dritten stattfindet.

Organisatorische Unterstützung eigenständiger Unterprojekte

Unterprojekte, also Projekte, die auf dem Open Source Programm aufsetzen, sind ein wichtiger Antrieb für Open Source Communities. Subgruppen bilden die Möglichkeit der Zielverfolgung im kleineren Kreise mit gleichgesinnten Entwicklern, ohne von einer großen Community abzuhängen. Bestes Beispiel hierfür sind Content Management Systeme, die ein Framework bieten, auf das einzelne Gruppen aufsetzen können, um spezifische Module für Individuallösungen zu erstellen. Unterstützt ein Projekt diesen Prozess, honoriert es die Arbeit an und mit der eigenen Software und erkennt so Mitglieder der Community an. Somit kann Anerkennung Kollaborateuren eines Open Source Projektes auch durch die organisatorische Unterstützung eigenständiger Unterprojekte zu teil werden. Mit 61% Vorkommnis in der Stichprobe dieser Arbeit handelt es sich auch hier um ein weitläufig genutztes Mittel zur Motivation durch Wertschätzung. Insbesondere ließ sich dieses Kriterium im Bereich der Content Management Systeme ausmachen. Alle analysierten Projekte in dieser Kategorie und damit 26,44% aller Projekte mit diesem Kriterium wiesen dieses Mittel aus
Dabei kann ein Open Source Projekt auf unterschiedliche Weise Subprojekte unterstützen. Die simple Erwähnung von Unterprojekten auf der Hauptseite etwa drückt Wertschätzung aus und weist Benutzer auf die Existenz jener hin.
Einen großen Schritt weiter geht die Bereitstellung von Ressourcen wie Kommunikationsplattformen und Kollaborationslösungen für die Subprojekte. Denkbar sind fertige Umgebungen, die sich neu gebildete Interessengruppen erstellen und dann nutzen können um an ihrem Projekt zu arbeiten, etwa ein Forum, eine Wiki oder Versionierungssysteme.

Noch größere Wertschätzung wird Entwicklern gegeben durch die finanzielle Subvention und Zuschüsse. Denkbar hier ist unter anderem die Bezahlung von für das Subprojekte essenziellen Diensten wie kostenpflichtige Server durch das Hauptprojekt. Wenn Entwickler erfahren, dass ihr jeweiliges Subprojekt und damit ihre Arbeit so wichtig ist, dass die Kosten der benötigten Ressourcen dafür vom Hauptprojekt getragen werden, wird ihnen große Anerkennung zu teil. Dies motiviert im hohen Maße und sorgt so direkt für eine langlebige Community.
Zahlreiche Subprojekte beeinflussen dabei ungemein die Langlebigkeit eines Projektes, denn wenn viele Gruppen auf das Framework als Basis vertrauen und zahlreiche Individuallösungen auf das Programm in Kombination mit der Erweiterung vertrauen, ist eine Fortsetzung des Projektes wahrscheinlich.

Aufstiegsmöglichkeiten

Auch erfahren Open Source Community-Anhänger Anerkennung durch Aufstiegsmöglichkeiten innerhalb einer Organisationsstruktur. Ein hierarchischer Aufbau, etwa mit einem Kernteam, welche die meisten Rechte hat, bietet potenziell Motivation durch Karrieremöglichkeiten innerhalb des Open Source Projektes. Wenn man sich durch eine bestimmte Position oder Stellung abheben kann und das auf der Seite durch eine Vorstellung des Kernteams auch noch nach außen getragen wird, fühlt man sich aus der Masse der Gesamtcommunity hervorgehoben und somit gewertschätzt. 61% der Stichproben-Projekte und damit eine beachtlich hohe Zahl der analysierten Open Source Projekte nutzen dieses Mittel.
Dabei kann zwischen verschiedenen Ansätzen unterschieden werden. Manche Projekte haben einen Fokus auf eine Oligarchie, also die Kontrolle von Wenigen mit geringen Ein- und Aufstiegschancen. In diesem Falle fühlt sich dieses Kernteam durch die besondere Stellung natürlich anerkannt, aber meist nur wenigen ist diese Wertschätzung und Motivation zu teil.
Eine Ausprägung, die mehr Menschen Anerkennung verleiht, ist eine Hierarchiestruktur mit mehreren Stufen. Somit wird einer breiteren Masse die Möglichkeit gegeben, sich innerhalb einer Organisation anerkannt zu fühlen und auch schon in geringeren Stufen mit eventuell noch nicht ausgeprägten Rechten eine Wertschätzung zu erfahren. Durch eine feinere Granularität kann dabei potenziell eine breitere Benutzerschicht erreicht werden, wenn etwa für die ersten Stufen nur eine Grundmitarbeit Voraussetzung ist. So fühlen sich auch Neulinge im jeweiligen Projekt schnell wertgeschätzt und motiviert.
Des Weiteren ist die Transparenz ein entscheidender Faktor: Wenn die Mitglieder des Projekts klar absehen können, wann sie in der Rangstufe aufsteigen und was sie dafür tun müssen, ist eine klare Definition gegeben. Dadurch kann der Aufwand abgeschätzt werden, um Anerkennung zu erreichen. Dies wiederum motiviert und schafft bessere Voraussetzungen für

die individuelle Anerkennung. Wenn dagegen scheinbar zufällig oder ohne ersichtlichen Grund eine Herab- oder Aufstufung einzelner Mitglieder erfolgt, ist weniger Anerkennung gegeben, da der Grund nicht ersichtlich ist und das System somit nicht respektiert werden kann.

4.5 Entwicklung eines Werkzeugs

Im vorherigen Kapitel wurden bereits die Anerkennungskriterien, welche auch in der nachfolgenden Tabelle gelistet werden, definiert. Die Einteilung der Anerkennungsausprägung in Grade und unterschiedliche Gewichtung dieser basiert auf der zuvor ausgeführten statistischen Analyse und den Erkenntnissen die aus der Recherche zu den einzelnen Projekten gewonnen werden konnte.

Die Ausprägung eines jeweiligen Kriteriums wird in Grade von 0-3 unterteilt. Grad 0 beschreibt dabei den Fall, dass das Kriterium in einer Entwicklungscommunity nicht vorhanden ist. Grad 1 setzt voraus, dass das Kriterium vorhanden ist, jedoch in einer geringen Ausprägung. Grad 2 impliziert nicht zwangsläufig Grad 1 jedoch ist die Ausprägung des Anerkennungskriteriums höher als bei Grad 2. Im Rahmen dieser Arbeit gipfelt die Ausprägung eines Anerkennungskriteriums in Grad 3. In der Tabelle sind die 5 Anerkennungskriterien und ihre jeweilige Grade gelistet.

		Grad der Ausprägung			
	Kriterium	**Grad 0**	**Grad 1**	**Grad 2**	**Grad 3**
1	Organisatorische Unterstützung eigenständiger Unterprojekte	nicht vorhanden	Namentliche Erwähnung des Unterprojekts auf der Webseite des Open Source Projekts	Angebot von Plattformen (Kollaborationslösungen) für das Unterprojekt	Subventionen/ Zuschüsse
2	finanzielle Kompensation pro Entwickler	nicht vorhanden	Ausschüttung	Bezahlte Addonsystem	Jobangebote
3	persönliche Vorstellung	nicht vorhanden	Nennung neuer Mitglieder	Organisations-Chart	detailliertes Org-Chart
4	Aufstiegsmöglichkeiten	nicht vorhanden	Zentrierung auf Oligarchie	existent, aber Hierarchiestruktur nicht voll ausgebildet/ intransparent	klar definierte, transparente Hierarchiestruktur
5	Auszeichnungen	nicht vorhanden	persönliche Auszeichnung in Form von Anerkennungsmail	Öffentliche Auszeichnung (Developer of the month)	langfristige öffentliche Würdigung erbrachter Leistung (Hall of fame)

Tabelle 4 Einteilung des Grads der Ausprägung des Anerkennungssystems

4.5.1 Organisatorischen Unterstützung eigenständiger Projekte

Dem Anerkennungskriterium der „**Organisatorischen Unterstützung eigenständiger Projekte**“ wurde als Grad 1 die *namentliche Erwähnung des Unterprojekts auf der Webseite des Open Source Projekts* zugeordnet. Grad 2 zu diesem Kriterium ist das *Angebot von Plattformen für das Unterprojekt* und Grad 3 sind *Subventionen für das Unterprojekt*.

Die ausschließliche Nennung des Unterprojekts sorgt für eine geringere Anerkennung, als das Anbieten einer Plattform für das Unterprojekt. Dies rührt daher, dass eine Erwähnung des Namens mit Verlinkung zwar prinzipiell das Interesse von Besuchern und Mitgliedern des Hauptprojektes auf das entsprechende Unterprojekt lenkt, es aber also nur durch eventuellen Neuzuwachs an Unterstützern subventioniert. Es findet also keine aktive Unterstützung statt. Der passive Hinweis auf das jeweilige Subprojekt ist jedoch auch eine Art offiziell ausgesprochene Tolleranz gegenüber abgeleiteten eigenständigen Projekten und somit eine Anerkennungsbekundung- auch wenn sie nur gering ist.

Das aktive Fördern der Unterprojekte innerhalb eines Projekts durch die Bereitstellung von Kollaborationstools etc. zeigt, dass das Unterprojekt aktiv gefördert und gewürdigt wird. Anders als nur bei einer namentlichen Nennung erfolgt hier direkte Unterstützung durch bereitgestellte Ressourcen.

Einen Schritt weiter dann werden nach dem Modell dieser Arbeit Projekte bewertet, die ihre eigenständigen Unterprojekte sogar durch finanzielle Mittel entschädigen, etwa in Form der Bezahlung von Monatskosten eines Servers. Monetäre Anerkennung zeugt nicht nur von außerordentlicher Wertschätzung, sondern hilft auch dabei, die Unterprojekte am Leben zu erhalten. Außerdem stärkt es die Reputation des Projektes.

Durchweg konnten alle Ausprägungen der Grade in unserer Analyse beobachtet werden. Beispielsweise ist das Kollaborationstool Foswiki Grad 1 zuzuordnen, da das Unterprojekt auf der Projektseite lediglich erwähnt wird. Das Web Framework Ruby dagegen fördert seine Unterprojekte sogar durch die Bereitstellung von Tools und Plattformen. Wie schon erwähnt, erreicht die organisatorische Unterstützung von Unterprojekten Grad 3, wenn die Subprojekte finanziell bezuschusst bzw. subventioniert werden. Dies ist z.B. der Fall bei der ECMA- Skript Bibliothek JQuery, wo die jeweiligen Unterprojekte finanzielle Aufwandsentschädigungen erhalten.

4.5.2 Finanzielle Kompensation an Entwickler

Bei dem Kriterium der „finanziellen Kompensation pro Entwickler“ geht es konkret um monetäre Anerkennung, die einem oder mehreren Entwicklern direkt zu Gute kommt. Grad 1 beschreibt in einem Open Source Projekt bzw. in der Entwicklungscommunity eines Projektes die Ausschüttung von finanziellen Mitteln. Grad 2 impliziert bezahlte Addonsysteme und Grad 3 ein Jobangebot, welches dem Entwickler offeriert wird.
Unter dem Grad 1 wird in dieser Arbeit die unregelmäßige Auszahlung finanzieller Mittel an Projektaktive verstanden. Erhält ein Projekt diese Ausschüttungen ist nicht klar, wann diese tatsächlich einem Entwickler zukommen. Diese Art der finanziellen Kompensation erfolgt meist ohne festen zeitlichen Rahmen durch überschüssige Spendengelder oder sonstige Projekteinnahmen. Erhält man als Projektaktiver solch eine Würdigung, ist die dadurch erlangte Motivation zwar hoch, aber sie erfolgt meist selten und ist somit unzuverlässlich. Dadurch trägt dieses Mittel zwar zur Wertschätzung der Community bei, kann jedoch nicht als wesentlich für die Motivation insbesonderer vieler aufgefasst werden. Auch sind solche individuellen Ausschüttungen meist nicht öffentlich, so dass keine Wirkung nach Außen erzielt werden kann und die öffentliche Anerkennung ausbleibt.
Verglichen dazu ist der 2. Grad, die Möglichkeit der Bereitstellung von Modulen und Erweiterungen gegen Geld, welches direkt dem jeweiligen Entwickler zukommt, mess- und mitunter auch zeitlich abschätzbar. Insbesondere Open Source Projekte, die ein Framework erstellen, bieten meist eigene Plattformen an auf denen Entwickler selbst geschriebene Erweiterungen anbieten, die auf das Grundgerüst des Hauptprojektes aufsetzen. Hier sind vor allem Content Management Systeme zu erwähnen, die auch im Rahmen dieser Arbeit analysiert wurden. Die geschriebenen Erweiterungen machen aus der Standardlösung des eigentlichen Grund-Open-Source-Projektes passende Individuallösungen und werden teilweise eigens für ein bestimmtes Unternehmen oder Vorhaben geschrieben. Diese sind bereit, für diese maßgeschneiderte Lösung den jeweiligen Entwickler zu entlohnen. Das eigentliche Grundprojekt stellt dabei nur die Grundsoftware und das Portal, in dem Entwickler und an Einzellösungen-Interessierte zusammenfinden. Durch diesen Prozess wird dem Entwickler ein hohes Maß an Wertschätzung zuteil. So merken sie direkt an der Bezahlung, dass ihr Programm benötigt wird und ihre Arbeit wichtig ist. Diese Möglichkeit der Anerkennung ist also als hoch motivierend einzustufen.
Darüber kommt als finanzieller Wertschätzungs-Prozess nur noch eines: Die Festanstellung. Schafft es ein Projekt, ihren aktiven Mitgliedern oder externen ein Jobangebot machen zu können und somit Vollzeitkräfte zu beschäftigen, wird das Projekt nicht länger etwa als

Hobby oder unbezahlte Nebenbeschäftigung verstanden sondern vielmehr als nachhaltig motivierendes und ernstzunehmendes Vorhaben. Als Vollzeitarbeiter an einem Open Source Projekt arbeiten zu können gibt die höchste Anerkennung noch über dem Prozess des bezahlten Add-On Systems, denn es ist nicht länger das Produkt, was bezahlt wird, sondern der Mensch an sich. Dies schafft die höchste monetäre Wertschätzung und ist somit Grad 3 in unserer Arbeit.

Beispiele für diesen Prozess konnten in unserer Analyse gefunden werden. Grad 2 wird z.B. bei dem Joomla Content Management System Projekt erreicht. Grad 3, also, dass ein einzelner Entwickler ein bezahlter Job für das Projekt angeboten werden kann, kommt beispielsweise im Content Management System Jahia vor.

4.5.3 Persönliche Vorstellung

Der erste Grad für das Anerkennungskriterium „**persönliche Vorstellung**“ ist die *Nennung neuer Mitglieder* auf der Projektseite. Grad 2 wird erreicht, wenn ein Organisations-Chart existiert. Grad 3 ist die logische Weiterführung von Grad 2, die Existenz eines *detaillierten Organisations-Charts*.

Die Nennung eines neuen Mitgliedes zeigt zwar Anerkennung, jedoch integriert dieses zunächst nicht in die Entwicklungscommunity und honoriert insbesondere keine spezifische Leistung. Die simple Nennung von jedem Neumitglied ist außerdem vollkommen unspezifisch, denn prinzipiell wird zunächst jeder in dieser Form “gewertschätzt”, ob er nun danach gute oder schlechte Arbeit leistet, ob er sich aktiv oder garnicht an dem Projekt beteiligt. Dieses Mittel ist somit nicht sonderlich motivierend und wirkt nur initial. Aus diesem Grund ist Anerkennung bei der Nennung eines Mitgliedes weniger ausgeprägt als bei Grad 2 oder Grad 3.

Um Anerkennung zu erfahren, müssen individuelle Eigenheiten der Mitglieder vorgestellt werden. Dies geschieht erst, sobald ein Mitglied in einem Organisations-Chart einer Rolle zugeordnet wird. Existiert ein detailliertes Organisations-Chart, welches die einzelnen Mitglieder namentlich und bestenfalls mit Steckbrief aufführt, zeigt dies, dass die Rolle, die ein Mitglied in einem Projekt hat, anerkannt wird. Diese Organisationstabellen und Aufstellungen sind öffentlich zugänglich und so kann auch von externen Besuchern jedes Mitglied und seine Rolle begutachtet werden. Das Open-Source-Projekt und die Open-

Source-Community ist damit offensichtlich stolz auf ihre Mitglieder und sorgt so für erhöhte Motivation. Dieser Prozess ist in dieser Arbeit als Grad 2 definiert.
Des Weiteren ist die Anerkennung um so glaubwürdiger, je individueller und ausführlicher sie ausfällt. Die simple Namensnennung und die dahinter stehende organisatorische Einheit alleine ist somit nicht so hoch zu bewerten als etwa ein ausführliches Profil mit Bild und dem Lebenslauf der vorgestellten Person. Der höchste Grad 3 dieses Kriteriums ist somit die persönliche Vorstellung von Community-Mitgliedern in einem detaillierten Organisationssystem. Je persönlicher und ausführlicher dieses System ist, desto mehr Anerkennung kommt jeder genannten Person zu gute. Dies sorgt folgerichtig für Motivation auch unter nicht erwähnten Mitgliedern, denn ihnen wird die Aussicht auf Nennung in diesem System gegeben.
Beispiele für dieses Kriterium konnten in unserer Analyse insbesondere für die Organisationsaufstellung gefunden werden. Bei dem Texteditor Notepad ++ existiert ein Organisations-Chart, welches die existierenden Rollen in dem Projekt aufzeigt. Dadurch erreicht dieses Projekt Grad 2 im Hinblick auf das Kriterium der persönlichen Vorstellung. Die Softwareentwicklungsumgebung Eclipse oder das linuxbasierte Betriebssystem Ubuntu erreichen durch ein detailliertes Organisations-Chart Grad 3.

4.5.4 Aufstiegsmöglichkeiten

Grad 1 des Anerkennungskriteriums „**Aufstiegsmöglichkeiten**“ beschreibt die *Zentrierung auf die Oligarchie*. Grad 2 erreicht ein Projekt, wenn eine Hierarchiestruktur existiert, jedoch intransparent ist bzw. nicht völlig ausgebildet. Grad 3 kann bei einer klar definierten, detaillierten Hierarchiestruktur erlangt werden.
Der erste Grad des Bewertungsmodells dieser Arbeit ist erfüllt durch den Zentrierungsprozess des zu untersuchenden Open Source Projekts auf eine Oligarchie. Dies drückt nach der griechischen Wortherkunft die Herrschaft durch wenige Privilegierte aus und äußert sich im Open Source Umfeld meist durch ein Kernteam mit maximalen Rechten, welche die Hauptverantwortung des Projektes tragen. Es ist dabei schwierig für externe Interessierte diesem Projekt beizutreten. Dadurch, dass bei dieser Fokussierung nur wenige Anerkennung erhalten und es zusätzliche für andere Mitglieder, besonders neue, schwierig ist, Teil der Oligarchie zu werden, ist die Anerkennung zwar vorhanden und für eine kleine Menge Aktiver auch hoch, jedoch auf die breite Masse gesehen gering zu

bewerten. Das Vorhandensein solch eines Prozesses ist in dieser Arbeit daher nur mit dem 1. Grad zu bewerten.

Um eine breitere Schicht an Benutzern und Interessierten anzusprechen bietet sich besonders ein Hierarchiesystem innerhalb des Open Source Projektes an. Dies bedeutet dass jedes aktive Mitglied eine Karriere innerhalb des Projekts geboten wird, angefangen mit kleineren Stufen, meist ausgestattet mit wenigen Rechten. Durch Mitarbeit oder Leistungen kann dann aufgestiegen werden und somit ein höherer Rang erreicht werden. Diese Möglichkeit des Aufstiegs ist auch im Zusammenhang mit dem Kriterium der "persönlichen Vorstellung" zu sehen, denn steigt man innerhalb der Organisationsstruktur auf, wird man auch durch die Darstellung der Laufbahn und der aktuellen Hierarchiestufe gewertschätzt. Aber auch ohne öffentliche Vorstellung gibt ein Hierarchiesystem individuelle Anerkennung. Durch die "Beförderung" wird einem Mitglied ebenso Wertschätzung zu teil als auch durch den Ausbau an Rechten innerhalb des Projektes. Grad 2 ist somit in der Bewertungsskala dieser Arbeit erreicht, wenn ein Hierarchiesystem vorhanden ist. Transparenz ist der entscheidende Punkt, der zum Erreichen des 3. Grades nötig ist. Eine klar definierte und detaillierte Hierarchiestruktur macht es für den einzelnen Entwickler ersichtlich, wann und vor allem wie er welchen Status in einem Projekt erreichen kann, wodurch er genau abschätzen kann, wie er weitere Anerkennung erreichen kann. Dies motiviert und regt an, noch aktiver mitzuarbeiten, um weitere Wertschätzung zu erhalten. Existiert also etwa eine Beschreibung oder Ähnliches, welche Stufen es in dem Projekt gibt und was getan werden muss, um diese zu erreichen, ist das Projekt mit Grad 3 zu bewerten.

Beispiele ließen sich in unserer Recherche für alle Kategorieausprägungen finden. Bei dem Open Source Projekt 7 Zip konnte eine Oligarchiestruktur festgestellt werden, was zu einer Bewertung 1. Grades führt. Das Kollaborationstool Openfire pflegt eine existierende Hierarchiestruktur, wodurch das Projekt Grad 2 erreicht. Bei dem Eclipse-Projekt gibt es eine klar definierte Hierarchiestruktur, somit erreicht es Grad 3 in dem System unserer Arbeit.

4.5.5 Auszeichnungen

Das fünfte Anerkennungskriterium sind die „**Auszeichnungen**“. Erhält der Entwickler eine *persönliche Auszeichnung* ist dies der erste Grad, der in einem Projekt erreicht werden kann. Die *öffentliche Auszeichnung,* ist durch die Darstellung nach außen mit Grad 2 zu bewerten. Grad 3 wird erreicht, wenn ein Entwickler nicht nur öffentlich, sondern *dauerhaft für erbrachte Leistung* ausgezeichnet wird.
Grad 1 in dem Bewertungssystem ist ein interner Prozess, in dem einem Mitglied eine persönliche Auszeichnung, etwa durch eine private Nachricht, verliehen wird. Dies ist ein Anerkennungsprozess und somit positiv zu bewerten, jedoch wird er nicht nach außen getragen. Andere Mitglieder und besonders externe Besucher sind von diesem Prozess ausgeschlossen und wissen dementsprechend nicht, dass durch aktive Mitarbeit an diesem Open Source Projekt Auszeichnungen erhalten werden können. Dies dämpft die Motivation und beschränkt den Effekt dieses Prozesses.
Mit Grad 2 und damit besser zu bewerten ist ein Open Source Projekt daher, wenn es die Auszeichnung öffentlich zur Schau stellt. Das Projekt ehrt den Entwickler vor Publikum, wodurch dieser Anerkennung vor einer größeren Öffentlichkeit erfährt als dies bei einer persönlichen Auszeichnung, wie z.B. einer Anerkennungsemail, der Fall ist. Die Wertschätzung ist sowohl für das einzelne Mitglied größer, denn er wird aus der Masse der Projektmitglieder hervorgehoben, als auch für andere Mitglieder, denn sie sehen, dass die Mitarbeit an diesem Projekt wertgeschätzt wird.
Wenn diese Auszeichnung sogar zeitlos gespeichert wird, also es sich nicht nur um Titel wie “Entwickler des Monats” handelt, ist die Motivation, diese noch bedeutendere und vor allem lange andauernde Ehre zu erhalten, noch Größer. Grad 3 in der Bewertungsskala dieser Arbeit ist somit für ein Projekt erfüllt, welches ein Prozess bietet, Mitglieder dauerhaft in einer Sektion des Projektes zu ehren und sogar zu verewigen. Dabei wäre es möglich, dass der Entwickler einen Platz in einer sogenannten “Hall of Fame” findet, welche ihn persistent auch nach dem Ausstieg aus einem Projekt ehrt.
Bei dem Content Management System Drupal ist ersichtlich, dass Entwickler z.B. als “Developer of the month” öffentlich ausgezeichnet werden. Drupal ist somit in Grad 2 des Auszeichnungskriteriums zu finden. Das Content Management System Xoops zeichnet Entwickler, welche eine besondere Leistung erbracht haben, aus indem sie diese in der “Hall of Fame” aufführen, was zu Grad 3 unseres Systems führt.

4.6 Netzdarstellung

Nachdem die Grade und deren Ausprägungen inhaltlich definiert wurden, soll im Folgenden gezeigt werden, wie ein Unternehmen methodisch ein Open Source Projekt im Hinblick auf die Grade leicht und übersichtlich analysieren kann. Zur Visualisierung wird ein Netz zur Anerkennungsevaluierung herangezogen, welches das Werkzeug für die Analyse eines Projektes darstellt. Neben dem Netz wird zusätzlich noch die Tabelle mit der Definition der Grade für eine vollständige Analyse eines Projektes benötigt.
Das Grundgerüst bzw. die Basis des Netzes stellt Abbildung 10 dar.

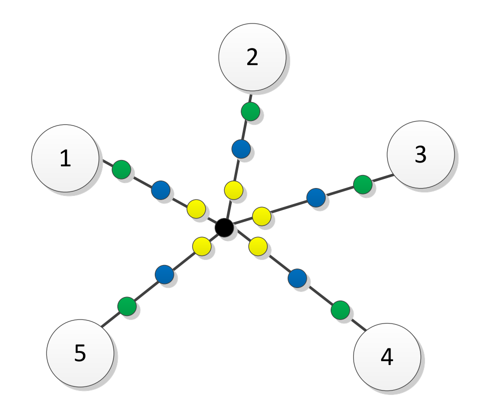


Abbildung 10 - Netzdarstellung Auswertung

Das gezeigt Grundgerüst setzt sich zunächst aus den äußeren 5 Hauptkreisen zusammen, die wiederum kreisförmig angeordnet sind. In diesen Kreisen ist jeweils eine Nummer von 1-5 zu finden, welche jeweils für ein zuvor definiertes Anerkennungskriterium stehen.

Dabei steht die Ziffer

1 für die **finanzielle Kompensation**,
2 für die **persönliche Vorstellung**,
3 für die **Auszeichnung**,
4 für die **organisatorische Unterstützung eigenständiger Unterprojekte** und
5 für die **Aufstiegsmöglichkeiten**.

Von jedem Kriterium verläuft eine Gerade zur von den Hauptkreisen umschließenden Mitte hin, wodurch eine Art Stern entsteht, wie Abbildung 10 zeigt. Auf den jeweiligen Geraden liegen wiederum 3 kleine Kreise. Diese stehen für die jeweiligen Grade der Anerkennungskriterien. Der grüne Kreis auf der jeweiligen gerade steht für Grad 3, sprich den höchsten zu erreichenden Grad. Der blaue Kreis zeigt jeweils Grad 2 und der gelbe Grad 1. In der umschließenden Mitte findet sich ein schwarzer Punkt, welcher Grad 0 und damit das nicht vorhanden sein eines Anerkennungskriteriums beschreibt.

In der Durchführung wird das zu analysierenden Projekt auf dessen Anerkennungskriterien und Ausprägungsgrade hin untersucht. Dies geschieht methodisch mit Hilfe des Netzes zur Anerkennungsevaluierung durch die Verbindung der Grade, entspricht den kleinen Kreisen, von den einzelnen Anerkennungskriterien, entspricht den 5 Hauptkreisen. Daraus ergibt sich dann ein Netz, welches zusammenfassend die Kriterien und deren Ausprägungen zeigt.

In Abbildung 13**Error! Reference source not found.Error! Reference source not found.** soll die zuvor beschriebene Vorgehensweise zur Anwendung des Netzes zur Anerkennungsevaluierung beispielhaft gezeigt werden.

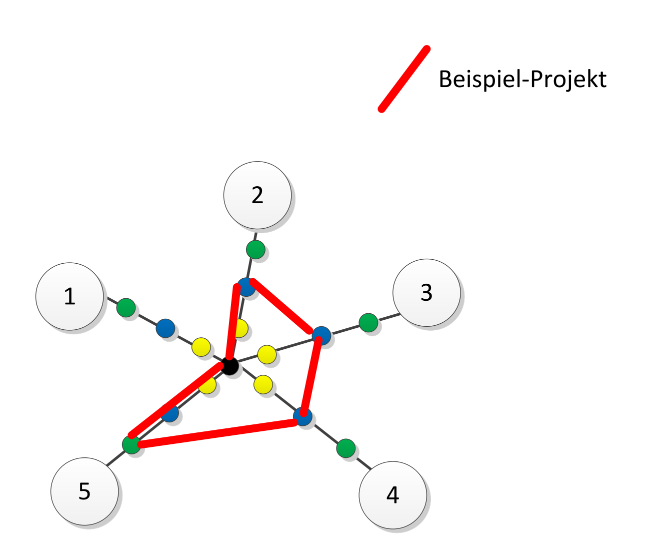


Abbildung 11 – Netzdarstellung Beispielprojekt

In Abbildung 13 entspricht das rote Netz dem zu untersuchenden Beispielprojekt. Beginnend bei Kriterium eins wird schnell ersichtlich, dass die **finanzielle Kompensation** in diesem Projekt nicht vorhanden ist (Grad 0). Weiterhin wird bei den Kriterien **persönliche Vorstellung**, **Auszeichnungen** und **organisatorische Unterstützung eigenständiger Unterprojekte** Grad 2 in dem Beispielprojekt erreicht. Das Kriterium **Aufstiegsmöglichkeiten** erhält Grad 3 in diesem Projekt. Wofür die einzelnen Grade stehen wurde bereits in Kapitel 0 definiert.

Diese Vorgehensweise kann nun für viele Projekte angewandt werden, wodurch eine Vielzahl an Netzen entstehen. Dadurch wird der Vergleich von den verschiedenen Projekten anschaulich abgebildet und vereinfacht. Dies erleichtert einem Unternehmen die Anerkennungsevaluation.

4.7 Praktische Anwendung

Aus kommerziell ausgerichteter unternehmerischer Sicht kann das zuvor entwickelte Werkzeug des Anerkennungsrasters unter zwei Zieleinsetzungen eingesetzt werden:

- Der Einschätzung eines OSS-Projekts hinsichtlich der Verwendbarkeit in der eigenen IT
- Der Adaption der Erfolgsfaktoren eines OSS-Projekts in der eigenen Organisation

Der erste Punkt betrifft die Einschätzung von OSS hinsichtlich ihrer Eignung für den Einsatz in einem kommerziellen Projekt, sei es integriert in proprietärer Softwareentwicklung oder eigenständig. Wie beispielsweise im Qualification and Selection of Open Source Software (QSOS) Modell dargestellt[24], ist es für ein Unternehmen entscheidend zu wissen ob ein OSS-Projekt industrielle Anforderungen hinsichtlich von Robustheit und **Beständigkeit** erfüllen kann. Aus Sicht traditionell organisierter IT ist es zudem interessant **von OSS-Projekten zu lernen**, da einige so großen Erfolg haben, dass sie kommerzielle Software nicht nur aus Kostengründen sondern auch auf Grund ihrer Qualität verdrängt haben. Beispiele hierfür sind der Firefox Browser und die Eclipse Softwareentwicklungsumgebung. Projekten dieser Güte wird uneingeschränktes Vertrauen geschenkt und sie erhalten oftmals Vorzug gegenüber kommerziellen Alternativen wie z.B. dem Microsoft Internet Explorer.

Das Anerkennungsraster kann z.B. durch die fachliche Leitung eines IT-Projekts in dessen Planungsphase verwendet werden. Es kann dabei die Entscheidung vereinfachen ob ein OS Softwareprodukt innerhalb konkreterer Planung berücksichtigt werden kann oder ob das generelle **Risiko** so hoch ist, dass der Fokus frühzeitig auf andere Produkte gerichtet werden sollte. In der Entwicklung der Oberfläche von Webanwendungen ist das Framework jQuery gegenwärtig sehr populär. Unabhängigkeit von seiner Marktdurchsetzung und Popularität muss das Risiko eingeschätzt werden, ob mit der Weiterführung des Projekts in mittelfristiger Sicht zu

[24] Vgl. o.V. (2006), http://www.qsos.org

rechnen ist. Die Beständigkeit eines OSS-Projekts ist wie mehrfach erwähnt mit der Mitarbeitermotivation verbunden.

Exemplarisch sei das Anerkennungsraster nachfolgend auf jQuery angewendet:

		Grad der Ausprägung			
	Kriterium	**Grad 0**	**Grad 1**	**Grad 2**	**Grad 3**
1	Organisatorische Unterstützung eigenständiger Unterprojekte				X
2	Finanzielle Kompensation pro Entwickler	X			
3	Persönliche Vorstellung				X
4	Aufstiegsmöglichkeiten				X
5	Auszeichnungen		X		

Tabelle 5 Anerkennungsraster jQuery

Das Projekt weist eine Vielzahl von Unterprojekten auf, die klar organisiert zu sein scheinen und übergreifend durch das jQuery Board zusammengeführt werden. Für Entwickler ist es zudem möglich finanzielle Unterstützung für ihre Arbeit zu beantragen. Daher lässt sich dem Kriterium 1 der Grad 3 zu ordnen. Über direkte finanzielle Kompensation pro Entwickler sind keine Informationen zu erhalten. jQuery ist daher für Kriterium 2 der Grad 0 zugeordnet. Die persönliche Vorstellung ist ausgeprägt, so werden das Führungsgremium sowie ein Kernteam von Entwicklern auf der Homepage des Projekts mit Foto ausgewiesen. Das ist gegenüber anderen Projekten überdurchschnittlich detailliert. Über das Kriterium Auszeichnungen konnten wenige Informationen erhalten werden. Daher wurde unter Vorbehalt der Grad 1 gewählt. Nach der Bewertung von jQuery durch dieses Raster können dessen Ergebnisse nun visuell in einem Netz abgebildet werden:

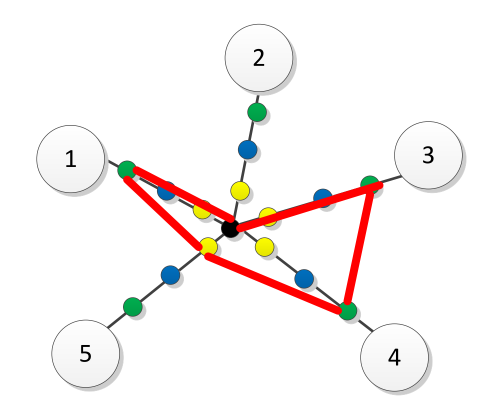


Abbildung 12 – Netzdarstellung jQuery

Wie in Abbildung 12 zu erkennen erhält jQuery die Mehrzahl der Kriterien in Grad 3. Somit schließen wir in Bezug auf die Erzeugung von Mitarbeitermotivation durch Anerkennungsmaßnahmen, dass das Produkt nicht mehr in Kinderschuhen steckt, sondern eine Reife erreicht hat, die es ihm ermöglichen sollte produktiv eingesetzt werden zu können.

Exemplarisch sei das Werkzeug nachfolgend ein weiteres Mal auf ein weiteres Projekt angewendet. Für das Projekt

		Grad der Ausprägung			
	Kriterium	**Grad 0**	**Grad 1**	**Grad 2**	**Grad 3**
1	Organisatorische Unterstützung eigenständiger Unterprojekte	X			
2	Finanzielle Kompensation pro Entwickler	X			
3	Persönliche Vorstellung		X		
4	Aufstiegsmöglichkeiten	X			
5	Auszeichnungen	X			

Tabelle 6 Anerkennungsraster GNU Hurd

Die **Anerkennungsmaßnahmen** von GNU Hurd scheinen **insgesamt nicht sehr ausgeprägt** zu sein. Weder eine außerordentliche organisatorische Unterstützung von Unterprojekten, finanzielle Kompensation pro Entwickler, Auszeichnungen noch die Aussicht des Aufstiegs in der Hierarchie des Projekts ist zu erkennen. Lediglich die Existenz von persönlichen Seiten der Entwickler auf der Homepage des Projekts lassen das Kriterium mit Grad 1 zu bewerten.

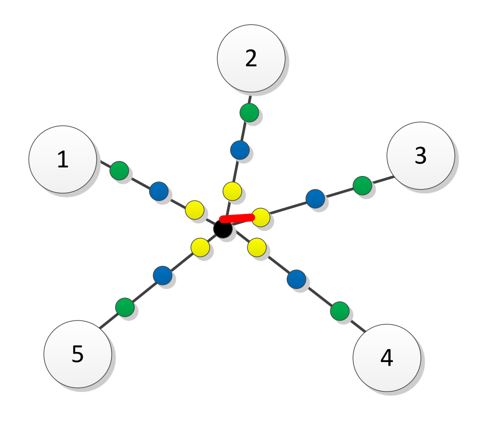


Abbildung 13 – Netzdarstellung GNU Hurd

Über die grafische Darstellung der Anerkennungsmaßnahmen von GNU Hurd in Abbildung 13 ist schnell zu erkennen, dass diese nicht sehr ausgeprägt zu sein schein. Es ist kein

ausgebildetes Netz zu erkennen. Aus dieser Sicht kann darauf geschlossen werden, dass der Einsatz des Produkts risikobehaftet ist und Alternativen evaluiert werden sollten.

Wie zu Beginn des Kapitels angesprochen zeigt diese Arbeit einen Weg auf wie Unternehmen von den Erfolgsfaktoren **von OSS-Projekten lernen** können. Die Ergebnisse dieser Untersuchung beinhalten welche Anerkennungsmaßnahmen in OSS-Projekten, z.B. in sehr Erfolgreichen, weit verbreitet sind.

Nachfolgende Grafik zeigt die Verbreitung der Anerkennungskriterien, die im Anerkennungsraster Anwendung gefunden haben.

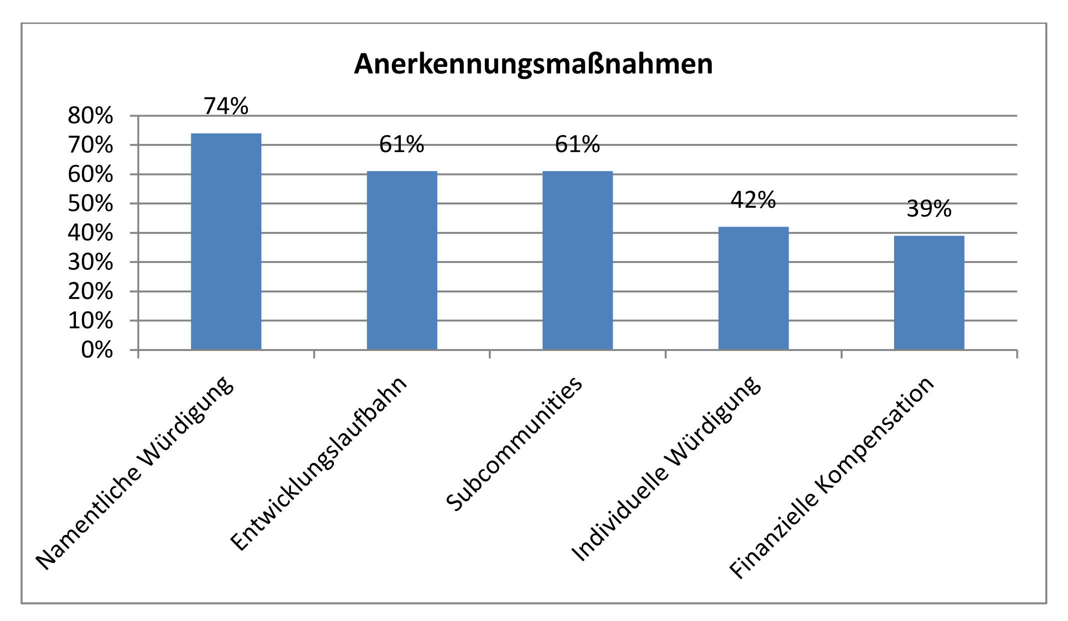


Abbildung 14 – Verbreitung der Anerkennungsmaßnahmen des Anerkennungsrasters

Wie in Abbildung 14 zu erkennen ist würdigen ca. 74% aller untersuchten Projekte die Leistungen ihrer Mitarbeiter durch eine öffentliche Nennung ihres Beitrags zum Community. Unternehmen kann somit z.B. empfohlen werden regelmäßig über E-Mails oder im Firmenintranet auf die Leistungen einzelner Projekte oder Abteilungen hinzuweisen. Ein geeigneter Platz könnte ferner auch Abteilungsmeeting sein. Es zeigt sich weiter, dass auch Wert darauf gelegt werden sollte, dass eine reale Chance besteht durch gute Leistungen in einer Fachlaufbahn aufzusteigen. Über die Hälfte der Projekte der Stichprobe (ca. 61%) und nahezu alle sehr erfolgreichen Projekte wie Ubuntu oder Mozilla implementiert eine Entwicklungslaufbahn. Sie zeigen klar auf welche Leistungen erbracht werden müssen um aufzusteigen und wer diese Entscheidung trifft. Erkennbar ist auch, dass Unternehmen ihre Mitarbeiter stets motivieren sollten eigene Ideen einzubringen und diese zu unterstützen.

Innerhalb ca. 61% der Stichprobe verfügten die Projekte über Subcommunities die nicht durch das Führungssystem entwickelt wurden.
Die explizite individuelle Würdigung eines Mitarbeiters konnte in ca. 42% der untersuchten Projekten gefunden werden. Innerhalb eines kommerziellen IT-Projekts sollte somit auch nachgedacht werden ob dieses für hervorragende Leistungen auf regelmäßiger Basis Mitarbeiter prämiert. Beispielsweise wie in ca. 39% durch finanzielle Leistungen.

Key Take-Aways

- 5 signifikante Anerkennungskriterien
  - ⇨ Konkretisierung durch Grad-System
  - ⇨ Visualisierung durch Netzdiagramme
  - ⇨ Bessere Vergleichbarkeit einzelner OS-Projekte

5 Grenzen der Arbeit

Die durchgeführte Untersuchung unterliegt beschränkenden Faktoren in Hinblick auf den organisatorischen Rahmen der Arbeit, als auch dem Umfang des bearbeiteten Themengebiets. Ein einschränkender organisatorischer Faktor sind die Ressourcen zur Durchführung der Analyse. Zum einen ist dies der zeitliche Rahmen der Arbeit, zum anderen die zur verfügbaren Personen.

Inhaltliche Grenzen zeigen sich in den Bereichen, in denen der Fokus von Anerkennungssystemen abweicht. Wie in Kapitel 1 und 2 dargestellt, konzentriert sich diese Arbeit auf den Zusammenhang zwischen Motivation und Anerkennung. Bei der Analyse von Anerkennungssystemen werden jedoch beispielsweise Themenbereiche gestreift, die zwar unmittelbar die Motivation zur Mitarbeit in OSS Projekten betreffen, jedoch keinen direkten Bezug zur Aussprache von Anerkennung erkennen lassen. Um die Arbeit fokussiert durchzuführen wurden daher nur unmittelbar relevante Aspekte vertieft. Dazu zählen nicht Themen wie Soziale Systeme, Führungsstil und Monetäre Vergütungssysteme, Corporate Identity oder Merchandise. Wie in Kapitel 2 dargestellt, lassen sich der Erfolg bzw. die nachhaltige Existenz eines OSS-Projekts über all diese Aspekte erklären. Sie sind miteinander verknüpft und können nur schwer losgelöst betrachtet werden. Gerade deswegen zeigt sich, dass es wahrscheinlich keine lineare Kette von Einflussgrößen gibt. Vielmehr ist es ein System aus Einflussgrößeren, zwischen denen mehr oder minder Große Wechselwirkungen bestehen.

Dies hat u.a. die Konsequenz, dass die Übertragung der Ergebnisse dieser Arbeit in ein, wie in Kapitel **Error! Reference source not found.** durchgeführt, praktisches anwendbares Werkzeug leicht die Komplexität des Systems ignorieren und zur Bildung von stärker subjektiv getrübten Schlussfolgerungen führen können. Die Erarbeitung eines Einteilungsrasters für OSS-Projekte unterlag der Herausforderung Abhängigkeiten zu priorisieren und weniger hoch priorisierte zu ignorieren.

Eine weitere Limitation der Arbeit ist die Beschränkung auf eine primär qualitative Auswertung der erhobenen Stichprobe in Kapitel 4. Eine hohe Anzahl von Projekten konnte nicht quantitative ausgewertet werden, dies ist jedoch relevant für weiterführende statistische Untersuchungen. Beispielsweise würde eine quantitative Analyse dazu beitragen die in Bezug auf den Zusammenhang von Anerkennungssystemen, Motivation und Erfolg eines OSS-Projekts aufgestellten Thesen dieser Arbeit zu erhärten.

Letztendlich konnte keine Anwendung der Ergebnisse dieser Arbeit auf eine weitere Stichprobe von OSS-Projekten durchgeführt werden. Ziel dieser Reflektion sollte es sein, die

herausgearbeiteten Anerkennungsfaktoren auf weitere Projekte, wie z.B. sehr erfolgreiche, wiederholt anzuwenden. Gerade im bearbeiten Bereich der Organisation von Projekten bzw. sozialen Systemen können Thesen am besten dadurch verifiziert werden, dass eine empirische Analyse mit möglichst großer Stichprobe durchgeführt wird. Diese trägt dazu bei die analytische Unschärfe zu mindern, die aus der Komplexität eben genannter Systeme resultiert und sich in den zuvor erwähnten inhaltlichen Grenzen der Analyse niederschlägt.

Überdies hinaus ist der generelle Eindruck der Untersuchung, dass es schwer ist ein umfassendes Verständnis über die Dynamik eines OSS-Projekts zu gewinnen, ohne dieses aus der Sicht eines Mitglieds von „innen“ zu sehen. Die Aussprache von Erkennung bzw. die Wertschätzung eines Projektmitglieds kann nur begrenzt über die in dieser Arbeit untersuchten äußerlich sichtbaren Merkmale erfasst werden. Um eine präzise Einschätzung geben zu können ist es von Nöten den Umgang innerhalb einer Community darüber real zu erfahren, dass eine direkte Mitarbeit in dieser erfolgt.

6 Fazit

Das Ziel dieser Arbeit lag darin, eine Untersuchung von Anerkennungssystemen im Kontext von OSS-Projekten durchzuführen. Dies hatte das Ziel über deren Zusammenhang zur Motivation von Mitarbeitern, einen Bezug zur Erklärung des Erfolgs bzw. Misserfolgs und der Dauer der Existenz eines Projekts herzustellen. Innerhalb gegenwärtiger Forschung wird gerade der Aspekt der Motivation als prägende Größe für die Entwicklung eines OSS-Projekts betrachtet. Im speziellen war es eine Aufgabe dieser Arbeit konkrete Anerkennungsmaßnahmen zu identifizieren und falls möglich zu priorisieren, sodass eine Werkzeug entwickelt werden kann, welches Unternehmen dabei unterstützt das Risiko durch den Einsatz eines OSS-Projekts besser einschätzen zu können.

Das primäre Ergebnis dieser Arbeit war die **Identifikation einer Schnittmenge von Anerkennungsmaßnahmen** zwischen den untersuchten Projekten. Generell muss festgestellt werden, dass das System der Einflussgrößen auf Mitarbeitermotivation eine Komplexität hat, die nicht über eine limitierte Anzahl von Abhängigkeiten erklärt werden kann. So konnten auch zwischen dem Ergebnis der Voruntersuchung bzw. Abgrenzung der Stichprobe und der eigentlichen Analyse keine statistisch feststellbaren Abhängigkeiten gefunden werden.

Vielmehr existiert ein komplexes System, innerhalb dessen diese Arbeit dazu beiträgt, den Ausschnitt des Einflusses von Wertschätzung auf Motivation innerhalb von OSS-Projekten besser zu verstehen. Gerade da die Ressource Humankapitel für diese eine so entscheidender Faktor ist, sollte die Signifikanz der Wertschätzung der Leistungen eines Mitarbeiters und der zugehörigen Aussprache von Anerkennung, innerhalb der Untersuchung von OSS eine nicht zu vernachlässigende Rolle spielen.

Anhang

Anhang 1 - Evaluation der Stichprobe ausgewählter OSS-Projekte 53
1.1 Kollaboration 53
1.2 Betriebssysteme 59
1.3 CMS 68
1.4 Administration 77
1.5 Büroanwendungen und Multimedia 83
1.6 Softwareentwicklung 88

Anhang 1 - Evaluation der Stichprobe ausgewählter OSS-Projekte

1.1 Kollaboration

Foswiki

Foswiki ist ein strukturiertes Wiki, welche in Perl programmiert wurde und als offene und programmierbare Kollaborationsplattform eingesetzt wird. Die Software kann betriebssystemunabhängig aufgesetzt werden. Das Projekt läuft unter der freien Software Lizenz GPL. Foswiki war ursprünglich Teil des TWiki Projekts. Mittlerweile stellt Foswiki eine Weiterentwicklung der TWiki Software dar. Der Gründer von TWiki, Peter Thoeny, hat versucht TWiki mit seiner Firma TWiki.NET immer mehr in die kommerzielle Richtung zu leiten. Damit konnte sich die Community nicht identifizieren und hat sich letztlich von TWiki gelöst. [25]

Kriterium	Vorhanden	Kommentar
Namentliche Würdigung von Leistungsträgern	Nicht erkennbar	
Events / Messen / Thementage	Ja	Event-Kalender existiert, z.B. Open Source World Conference, in verschiedenen Ländern/Städten
Honorar	Nicht erkennbar	
Wettbewerbe		
Entwicklungslaufbahn		
Grad der Nutzung der Kommunikationskanälen (Twitter, UserGroups)		
Feature of the month		
Developer of the month		
Taskteam		- Sub-Projekt bildung innerhalb des Foswikiprojekts gewünscht, Entwickler sind dazu aufgefordert andere Entwickler zu rekrutieren und an einem Sub-Projekt zu

[25] Vgl. FosWiki (o.J.), http://foswiki.org/About/WhyThisFork?redirectedfrom=Home.WhyThisFork

	arbeiten

Bei Foswiki können einige wenige Anerkennungskriterien gefunden werden. In der Community wird viel Wert auf Events gelegt bei denen die aktiven Mitglieder die Möglichkeit haben repräsentativ für das Software Projekt Foswiki öffentlich zu sprechen, wodurch diese Anerkennung von Personen aus anderen Open Source Projekten erfahren. Zusätzlich hat der Foswiki-Repräsentant die Möglichkeit seinen Horizont durch Gespräche mit Entwicklern aus anderen Projekten zu erweitern, wodurch neue Ideen für das eigene Projekt entstehen.[26] Weiterhin ist die Bildung von Subprojekten innerhalb des Foswiki-Projekts erwünscht. Die Entwickler werden dazu aufgefordert andere Entwickler zu rekrutieren, um zusammen an einem Teilprojekt von Foswiki zu arbeiten. Dadurch werden die Entwickler dabei unterstützt in selbstorganisierten kleinen Gruppen an einem Teilprojekt zu arbeiten, welches am besten ihre Interessen trifft.

[26] Vgl. Foswiki, (o.J.) http://foswiki.org/Community/ProjectCulture

Kunagi

Kunagi ist eine Webanwendung für das agile Projektmanagement nach der Scrum-Methode und eignet sich insbesondere für das Management von Softwareprojekten. Es zählt zu der Kategorie der Kollaborationstools[27]. Kunagi wird unter einer freien Software Lizenz (naheliegend GPL) vertrieben. Das Projekt von einem Entwicklerteam in Deutschland entwickelt.

Kriterium	Vorhanden	Kommentar
Namentliche Würdigung von Leistungsträgern	Ja	
Events / Messen / Thementage	Nicht erkennbar	
Honorar		
Wettbewerbe		
Entwicklungslaufbahn	Ja	- siehe unten: div. Kategorien (Product Owner, Scrum Master, ...)
Grad der Nutzung der Kommunikationskanälen (Twitter, UserGroups)	Nicht erkennbar	
Feature of the month		
Developer of the month		

Kungai gehört, wie Foswiki auch, zu den kleiner Projekten im Bereich Kollaboration. Weshalb auch in diesem Projekt Anerkennungskriterien kaum auffindbar sind. Eine Community Struktur ist bei Kunagi ausführlich dokumentiert. Die einzelnen Rollen und Aufgabenbereiche sind bei Kunagi klar definiert. Das Projektteam stellt sich aus 5 Personen zusammen. Kunagi hat eine weit verbreitete Webpräsenz auf verschiedenen sozialen Netzwerken, wie Facebook oder Twitter z.B. Öffentlich wird auf den genannten Plattformen Teammitgliedern für erbrachte Leistung gedankt. Diese Art von öffentlicher Anerkennung schafft Motivation für das Projekt und die Arbeit an diesem.

[27] Vgl. Source forge (o.J.), http://sourceforge.net/

Thunderbird

Thunderbird ist ein E-Mail Programm und Newsreader und gehört zu der Gruppe der Kollaborationstools. Diese Software ist ein Teil des Mozilla Projekts. Thunderbird bietet Mehrfachlizenzierung. Dabei ist Thunderbird für den Anwender mit einer GPL, LGPL und MPL Software Lizenz zu erwerbe. Jede dieser Lizenzen ist eine Open Source Software Lizenz.

Kriterium	Vorhanden	Kommentar
Namentliche Würdigung von Leistungsträgern	Nicht erkennbar	
Events / Messen / Thementage	Ja	-Themenparty um ein neues Produkt vorzustellen - Ein Grund, weshalb der Developer eine Adresse angeben muss; für Marketingzwecke
Honorar	Nicht erkennbar	
Wettbewerbe		
Entwicklungslaufbahn		
Grad der Nutzung der Kommunikationskanälen (Twitter, UserGroups)		
Feature of the month		
Developer of the month		

Insgesamt ist es schwer einen genaueren Einblick in die Community Struktur von Thunderbird zu erhalten, da sich Entwickler aus Gründen der Qualitätssicherung zunächst bewerben müssen um an dem Projekt teilzuhaben. Im Hinblick auf Anerkennung werden treten bestimmte Entwickler bei Events auf, um neue Produktversionen öffentlich vorzustellen. Zudem werden jährliche Feiern organisiert, zum Dank an die Mitwirkenden an dem Projekt. Mozilla bietet bezahlte Jobs für Entwickler an. Dies trägt maßgeblich zu der Anerkennung eines Einzelnen bei.

Pidgin

Die Software Pidgin, ursprünglich Gaim, ist ein Open Source Projekt, welches unter der freien GPL (General Public License) Software Lizenz läuft und in die Kategorie Kollaboration einzuordnen ist. Pidgin ist ein freier Multi-Protokoll Client, welche zunächst für unixähnliche

Systeme von Mark Spencer geschrieben wurde, aber mittlerweile auch auf Microsoft Windows läuft. Ein Multi-Protokoll Client beschreibt eine Software, welche mehrere Instant- Messaging Protokolle unterstützt[28]. Die Besonderheit ist, dass User sich mit dem Anmelden bei Pidgin gleichzeigt bei ihren verschiedenen Chat Networks angemeldet sind. Dadurch hat der User die Möglichkeit alle Chatfreunde aus verschiedenen sozialen Netzwerken in einem Chat Programm zu vereinen. [29]

Kriterium	Vorhanden	Kommentar
Namentliche Würdigung von Leistungsträgern	Nicht erkennbar	
Events / Messen / Thementage		
Honorar		
Wettbewerbe		
Entwicklungslaufbahn		
Grad der Nutzung der Kommunikationskanälen (Twitter, UserGroups)		
Feature of the month		
Developer of the month		

Die Analyse im Hinblick auf Anerkennung hat bei Pidgin ergeben, dass keine Anerkennungskriterien gefunden werden konnten. Aus der Recherche wurde nicht ersichtlich, ob bzw. welche Mittel zur Anerkennungsförderung existent sind. Allerdings ist dies kein Indikator für ein wenig erfolgreiches Projekt, da eine zu hohe Intransparenz zu den Vorgängen innerhalb der Entwicklungscommunity herrscht. Ein Nicht-Mitglied des Pidgin-Projekts erhält ohne eine Anmeldung kein Einblick die Community.

[28] Vgl. O.V. (o.J.), http://www.pc-magazin.de/ratgeber/multi-protokoll-client-1048848.html

[29] Vgl. Egan, S. (2005)

Openfire

Openfire ist ein XMPP- Server, welcher in Java geschrieben wurde und unter einer freien Open Source Apache Lizenz veröffentlicht wird. XMPP folgt dem XML-Standard und wird vorwiegend im Bereich Instant Messaging genutzt[30]. So gehört Openfire ebenfalls zu den Kollaborationsprojekten. Die Entwickler von Openfire sind Mitglieder des Softwareherstellers Jive.[31]

Kriterium	Vorhanden	Kommentar
Namentliche Würdigung von Leistungsträgern	Ja	- jeder Entwickler erhält Punkte für erbrachte Leistungen - Developer profile: Jeder Entwickler hat ein eigenes Profil mit Bild, Entwickler seit…, letzte Aktivität am Projekt - Rating / Kommentarsystem: Dokumente von einem Entwickler können von anderen Entwicklern bewertet und kommentiert werden
Events / Messen / Thementage	Ja	- Product release --> neue Versionen
Honorar	Nicht erkennbar	
Wettbewerbe		
Entwicklungslaufbahn	Ja	- je mehr sich ein Entwickler engagiert, umso höher kann dieser aufsteigen bis hin zum key contributer
Grad der Nutzung der Kommunikationskanälen (Twitter, UserGroups)	Nicht erkennbar	
Feature of the month		
Developer of the month	Ja	die Top Entwickler werden gelistet
Freundesystem		Innerhalb der Community gibt es ein Freundesystem

Bei Openfire sind Anerkennung und ein persönlicher und sozialer Umgang wichtige Aspekte in der Community. Entwickler werden für erbrachte Leistung namentlich genannt und erhalten Punkte. Über die Summe der Punkte, die ein Entwickler durch die Arbeit am Projekt gesammelt

[30] Vgl. o.V. (2003), http://searchdomino.techtarget.com/definition/XMPP

[31] Vgl. Jive Software (o.J.), http://www.igniterealtime.org/projects/openfire/

hat, definiert sich dessen Entwicklungslaufbahn. Darunter wird eine Hierarchie verstanden in der ein Entwickler vom Beginner zu einem Hauptakteur deklariert wird. [32]

Meinungen der einzelnen Community Mitglieder werden anerkannt, da die Community besonders Wert auf konstruktives Feedback legt und dieses auch umsetzt. Zusätzlich werden die Hauptakteure in dem Projekt namentlich genannt und ausgezeichnet in Form von „Developer oft the month“ bzw. „Most active developer“. Es werden organisatorische Subgruppen für die Bearbeitung von Teilprojekten (z.B. Features) innerhalb des Openfire Projekts gebildet. Des Weiteren schreibt der Projektführer regelmäßige Dankschreibungen an die gesamte Community für erbrachte Leistungen. Insgesamt sind die Anerkennungsmethodiken bei Openfire stark ausgeprägt.

1.2 Betriebssysteme

JNode

JNode ist ein Betriebssystem, welches die Besonderheit aufweist, außer auf Maschinensprache (Assembler) ausschließlich auf der Programmiersprache Java zu basieren. Aus mehreren Vorläufern entstanden zeigte sich innerhalb des Projekts einige Aktivität (zu sehen an Releases). Zeitlich weiter zurückliegende „neueste“ Forenbeiträge weisen jedoch auf eine momentane Inaktivität hin. Im Folgenden wird evaluiert, inwieweit die Community ihren Anteil an einem möglichen Projektscheitern hat.

<table>
<tr><th>Kriterium</th><th>Vorhanden</th><th>Kommentar</th></tr>
<tr><td>Namentliche Würdigung von Leistungsträgern</td><td>Ja</td><td>- Mitglieder und Führungspersonen sind namentlich gelistet</td></tr>
<tr><td>Events / Messen / Thementage</td><td rowspan="3">Nicht erkennbar</td><td></td></tr>
<tr><td>Honorar</td><td></td></tr>
<tr><td>Wettbewerbe</td><td></td></tr>
<tr><td>Entwicklungslaufbahn</td><td>Ja</td><td>- Entwicklung in Abhängigkeit von Grad und Qualität der Mitarbeit, sukzessive Erweiterung der Befugnisse</td></tr>
<tr><td>Grad der Nutzung der Kommunikationskanälen</td><td>Nein</td><td>- IRC
- Technical mailing list (SVN commit</td></tr>
</table>

[32] Vgl. Jive Software (o.J.), http://community.igniterealtime.org/community/developers/openfire_dev

(Twitter, UserGroups)		notification), forum
Feature of the month	Nicht erkennbar	
Developer of the month	Nicht erkennbar	

Das JNode weist insgesamt wenige Anerkennungssysteme auf, die Mitglieder zur Beteiligung am Projekt motivieren könnten. Die Community ist sehr klein, weswegen Beiträge/Leistungen zwar schnell bekannt werden, jedoch nicht in anderen Form „honoriert“ werden. Ähnlich verhält es sich mit dem Kriterium der Entwicklungslaufbahn: Auf der einen Seite müssen erst kleinere Beiträge eingereicht und für gut befunden werden, bevor ein Mitglied Zugriff auf das komplette Code-Repository erhält, andererseits entsteht hierdurch keine erkennbare Entwicklungslaufbahn. Einmal im Projekt existieren also für einen engagierten Entwickler keinerlei Möglichkeiten, sich einen privilegierten Status zu erarbeiten. Ebenso werden keine Beiträge oder Personen mit besonderer Auszeichnung gewürdigt und erhalten so Anreize, sich verstärkt zu engagieren und den Fortbestand des Projekts zu sichern.
Das Nicht-Vorhandensein einiger Anerkennungssysteme innerhalb der Community hat sicherlich zum ausbleibenden Erfolg des Projekts beigetragen. Allerdings könnten technische Limitationen bzw. Faktoren, die im Rahmen dieser Arbeit nicht beleuchtet wurden, auch durch eine „perfekte“ Community nicht ausgeglichen werden. Aus diesem Grund soll nicht der Eindruck erweckt werden, allein die Mitgliederbasis hätte den aktuellen Zustand des Projekts verantworten.

GNU Hurd

GNU Hurd ist ein Betriebssystem, dessen rein Open Source Community-basierte Entwicklung schon in den 1990er Jahren begann und bis heute andauert. Ursprünglich als Unix-Ersatz gedacht wurde für die Software auf technischer Ebene eine komplett neue Basis verwendet. Durch die Neugestaltung aller Kernkomponenten kam das Betriebssystem nie vollständig produktiv zum Einsatz. Einzelne Komponenten jedoch sind in späteren Linux-Distributionen zu finden. Inzwischen wurde eine Portierung des Hurd-Programmkerns auf andere Open Source Systeme auf Linux-Basis (z.B. Debian GNU/Hurd) vorgenommen, die jedoch noch nicht vollständig abgeschlossen ist. [33]

[33] Vgl. Debian GNU (2011), http://www.debian.org/ports/hurd/

Kriterium	Vorhanden	Kommentar
Namentliche Würdigung von Leistungsträgern	Ja	-Einzelne User können eigene Page aufbauen (zur Vorstellung + Arbeitsthemen)
Events / Messen / Thementage	Ja	- Google Summer of Code (GSoC) auch weitere, mit Hurd dev attendance vorhanden.
Honorar	Nein	- http://www.pro-linux.de/artikel/2/408/interview-mit-marcus-brinkmann.html
Wettbewerbe	Nicht erkennbar	
Entwicklungslaufbahn	Nicht erkennbar	
Grad der Nutzung der Kommunikationskanälen (Twitter, UserGroups)	Nein	- MailingLists, IRC, weblogs einzelner Entwickler
Feature of the month	Nicht erkennbar	
Developer of the month	Nicht erkennbar	

Im GNU Hurd Projekt sind nur sehr wenige Faktoren erkennbar, die Anerkennung für die Mitglieder der Community bieten. So existieren (zumindest nicht öffentlich) Informationen über Mitgliedertreffen bzw. Messen, auf denen das Projekt präsentiert wird und einzelne Mitglieder ihre Leistung herausstellen können. Auch wird, zumindest seitens des Hauptprojekts (für Debian GNU/Hurd ungesichert) kein Honorar geboten, welches Entwickler zur Mitwirkung motivieren könnte. Eine Entwicklungslaufbahn ist zumindest nicht dokumentiert, was allerdings nicht auf das Nicht-Vorhandensein schließen lässt: Anzunehmen ist, dass erst eine Mitwirkung erfolgen muss, damit Prozesse verdeutlich werden. In Bezug auf Anerkennung bietet GNU Hurd als Merkmal einzig und allein die Möglichkeit für Mitglieder, sich auf einer eigenen Sub-Website zu präsentieren.[34] Dies erzeugt Visibilität in der Community und jeder ist somit selbst dafür verantwortlich, andere über seine aktuellen Projekte zu informieren.

Ob insgesamt allerdings das Fehlen einiger als wichtig eingeschätzter Faktoren einen Rückschluss auf das Scheitern des Projekts zulässt, ist nicht gesichert. In diesem speziellen Fall führen sicherlich auch technische Aspekte dazu, dass die Software nie Marktreife erlangte. Zusätzlich dazu ist es gut möglich, dass diverse Anerkennungssystem erst deutlich werden, sobald eine Mitgliedschaft in der Community gegeben ist.

[34] Vgl. Debian GNU (2011), http://www.gnu.org/software/hurd/user.html

Arch Linux

Arch Linux ist eine Linux-Distribution, die auf grafische Installations- und Konfigurationshilfen zu Gunsten der Einfachheit verzichtet. Aus diesem Grund wird die Software eher für fortgeschrittene Anwender empfohlen. Seit dem ersten Release im Jahr 2002 wurde das Projekt kontinuierlich von der Community weiterentwickelt und zählt momentan zu den bekanntesten Linux-basierten Betriebssystemen.[35] Es existieren diverse Derivate der Software, die sich Arch Linux Komponenten zu Eigen machen und so eine abgewandelte Lösung kreieren (dank der GPL problemlos möglich).

Im Vergleich zu anderen großen OSS Betriebssystemen steht bei Arch Linux kein finanzkräftiges Unternehmen im Hintergrund, dass das Projekt fördert bzw. in eine gewisse Richtung lenkt. Nichtsdestotrotz wird überprüft, welche Aspekte die Community zum Erfolg des Gesamtprojekts beitragen lassen.

[35] Vgl. DistroWatch.com (2004), http://distrowatch.com/dwres.php?resource=popularity

Kriterium	Vorhanden	Kommentar
Namentliche Würdigung von Leistungsträgern	Ja	- "Nennung des Developers auf eigener Seite, die ausschließlich den Devs gewidmet ist: http://www.archlinux.org/developers/ same for trusted users: http://www.archlinux.org/trustedusers/" - "Neues aus dem Team" + "Wochenrückblick"
Events / Messen / Thementage	Ja	- Auf Messen anwesend, keine eigenen Thementage/Konferenzen ersichtlich: https://wiki.archlinux.org/index.php/DeveloperWiki:Linux_Conferences
Honorar	Nein	- Nennung des Developers auf eigener Seite, die ausschließlich den Devs gewidmet ist: http://www.archlinux.org/developers/ same for trusted users: http://www.archlinux.org/trustedusers/
Wettbewerbe	Nicht erkennbar	
Entwicklungslaufbahn	Ja	- dev by invitation only, versch. Stufen - e.g. "Trusted User" im AUR eigener "Status" bzw. KAtegorie im Forum (Member, Developer, ...)
Grad der Nutzung der Kommunikationskanälen (Twitter, UserGroups)	Ja	- Mailing Lists, IRC, Forum, Wiki, RSS Feeds anscheinend keine social Media präsenz
Feature of the month	Nicht erkennbar	
Developer of the month	Nicht erkennbar	
Platz für eigene Projekte	Ja	

Bei einer Analyse der Arch Linux Community wird stark deutlich, dass die entwickelte Software vor allem für fortgeschrittene Anwender konzipiert ist: Weder werden Grundverhaltensweisen im Sinne eines „Code of Conduct“ definiert, noch ist erkennbar, dass neuen Entwicklern ein Mentor zur Seite gestellt wird. Das Projektwiki selbst ist gut strukturiert und bietet eine Fülle an Informationen, jedoch sind diese ggf. für Neulinge schwer verständlich. In Bezug auf die Kommunikation verlässt sich das Projekt auf traditionelle OS-Entwicklungskanäle wie IRC, Mailing Lists und Foren. Auf soziale Netzwerke, die direkt dem Projekt zugeordnet werden können, wird verzichtet. Hierdurch entsteht also keine zusätzliche, öffentliche Präsenz und damit

Anerkennung außerhalb der internen Community. Bei Interesse ist jedoch eine Vorstellung einzelner (privilegierter) Mitglieder im Wiki zu finden. Hierdurch entsteht bspw. für Mitglieder des Developer-Teams oder für sog. „Trusted User“ ein Anerkennungssystem. Dieses ist auch durch die Existenz von verschiedenen Hierarchiestufen , speziell bei den Entwickler, gegeben, die einen Anreiz zum Aufstieg und (nach dem Erreichen eines bestimmten Rangs) Anerkennung bieten.

Auffällig war, im Vergleich zu anderen bekannten OS Betriebssystemen, dass im Wiki die Seriosität der Beiträge nicht immer gegeben war. So spricht ein Beitrag im „DeveloperWiki“ von „World Domination Plans“. Selbst wenn eine solche Aussage grundsätzlich als eher fragwürdig zu bewerten ist, zeugt sie doch vom Humor und Spaß, mit dem innerhalb des Projektes entwickelt wird. Dieser Faktor, sofern bekannt, ist zwar in Bezug auf die Anerkennung irrelevant, kann jedoch als weiteres Indiz für einen erfolgreichen Fortbestand der Community gewertet werden.

Fedora

Fedora ist ein Linux-basiertes Betriebssystem, welches aus dem ehemaligen Red Hat Linux entstanden ist. Allerdings erfolgte keine Umwandlung des Projekts in Open Source, stattdessen wurden die zugrundeliegenden Prinzipien von Beginn an eingehalten. Die Software wird, außer von der Community, aktiv vom Unternehmen Red Hat mitentwickelt. Die Lenkung des Projekts allerdings befindet sich in Händen der Community, da Red Hat seinen mehrheitlichen Stimmanteil im Fedora-Vorstand im Mai 2008 aufgab.

Statistiken bewerten diese Linux-Distribution als die momentan dritt-erfolgreichste nach Mint und Ubuntu.[36]

Kriterium	Vorhanden	Kommentar
Namentliche Würdigung von Leistungsträgern	Ja	- nicht messbar, user (contributor) werden irgendwann aufgrund ihrer Leistungen als "valuable" bewertet und können durch support in der Laufbahn aufsteigen
Events / Messen / Thementage	Ja	- Teilnahme auf Events/Messen, Eigene Messen für die Community (bspw. Release Parties),

[36] Vgl. DistroWatch.com (2004), http://distrowatch.com/dwres.php?resource=popularity

Honorar	Ja	- für bei RedHat festangestellte Entwickler
Wettbewerbe	Ja	- auf Messen und Großveranstaltungen
Entwicklungslaufbahn	Nicht erkennbar	- aber muss eigentlich vorhanden sein. verschiedene Rollen möglich (Devs, Art, BugFixes, Admins, Comms, ...)
Grad der Nutzung der Kommunikationskanälen (Twitter, UserGroups)		
Feature of the month		
Developer of the month		

In der Fedora-Community tragen eine Reihe von Faktoren zum Erfolg des Projekts bei. Trotz der Unterstützung durch das Unternehmen Red Hat beweist die Community durch eine klare Struktur, dass sie sich selbst organisieren kann. Dies wird auch deutlich durch die Abgabe der Stimmmehrheit im Vorstand durch Red Hat. Hierdurch erfährt die Community als solche Anerkennung, wobei sich jeder einzelne Mitwirkende angesprochen fühlen darf. Durch Messen, andere Events und Wettbewerbe wird der Zusammenhalt der Entwickler gestärkt. Ebenso ergibt sich hierdurch eine mögliche Präsenz und Anerkennung (durch Gespräche, Fachwissen) außerhalb des eigenen Kernkompetenzfelds. Persönliche Leistungen selbst werden nicht transparent nach außen hervorgehoben. Im Rahmen dieser Arbeit kann allerdings keine Aussage darüber getroffen werden, ob innerhalb der Community Anerkennungssysteme für Leistungen einzelner vorhanden sind.

Der Einstieg in das Fedora-Projekt gestaltet sich schwieriger als in anderen OSS Projekten: So ist bspw. kein Mentoring erkennbar, das Novizen Hilfestellung bieten könnte.[37] Auch ist das Konzept einer Entwicklungslaufbahn zwar vorhanden, allerdings bis jetzt nur als undokumentierter, inoffizieller Prozess. Diese mangelnde Transparenz sorgt für eine nur schwach existente Anerkennung nach außen. Innerhalb des Projekts werden Leistungen von Nutzern durch andere, erfahrenere Community-Mitglieder bewertet. Im Rahmen dessen erfahren Beiträge einer Wertschätzung. Ein „Aufstieg“ in der Entwickler-Hierarchie (in Bezug auf Zugriffs- bzw. Beitragsrechte) ist ab einem gewissen Level nur durch persönliches Sponsoring durch einen höheren Entwickler möglich. Dies zwingt dazu, ein fundiertes Netzwerk aufzubauen und zu pflegen, wodurch die Bekanntheit eines Mitglieds innerhalb der Community gesteigert wird.

[37] Vgl. Thoma, J. (2011), http://www.golem.de/1102/81153.html

Reichhaltige Kommunikationskanäle (klassische Medien wie Mailing-Listen, User Groups oder IRC-Kanäle oder aber Gemeinschaften in sozialen Netzwerken wie Facebook, Twitter, Xing, etc.) sorgen für eine hohe Visibilität und Bekanntheit anderen Nutzern gegenüber. Dies wird auch durch Informationen über die Community gefördert: Nicht nur weiß das Mitglied, wie die „Core Values“ des Fedora-Projekts lauten, sondern es existiert auch die Möglichkeit, sich über die Nutzerbasis als solche (Größe etc.) und über einzelne andere Mitglieder mit Hilfe personalisierter Wiki-Seiten zu informieren.

Ubuntu

Ubuntu ist ein kostenloses, Linux-basiertes Betriebssystem, welches seit 2004 existiert. Es wurde auf Basis der „Debian“-Linux-Distribution erstellt. Das Projekt wird vom Unternehmen *Canonical Ltd.* gesponsert, das vom südafrikanischen Unternehmer Mark Shuttleworth gegründet wurde.[38] Im Falle einer Aufkündigung der Projektweiterentwicklung seitens Canonical existiert die „Ubuntu Foundation“, die die langfristige Weiterentwicklung und Unterstützung der Software sicherstellen soll.[39] Auf funktionaler Ebene stellt die Distribution mit dem Ziel der Benutzerfreundlichkeit ein einziges Programm pro Anwendungsfall zur Verfügung. Nichtsdestotrotz wird dem Nutzer die Möglichkeit gegeben, seine Auswahl selbst zu treffen – die Vorabinstallation erleichtert ausschließlich den Einstieg in das Betriebssystem und sorgt für eine gute Abstimmung der Software-Komponenten untereinander. Bereits 6 Monate nach Erscheinens des ersten Releases war Ubuntu zur meistinstallierten Linux-Distribution auf Heim-PCs geworden.[40] Auch heute noch ist das Projekt (basierend auf der Anzahl der Website-Zugriffe) eines der bekanntesten Linux-Betriebssysteme.[41] Im Folgenden wird evaluiert, ob Anerkennungssystem innerhalb der Ubuntu-Community ersichtlich werden, die ihren Teil zum rapiden Wachstum und Fortbestand des Projekts beitragen.

[38] Vgl. The Ubuntu Foundation (2011a), http://www.ubuntu.com/project/canonical-and-ubuntu

[39] Vgl. The Ubuntu Foundation (2011b), http://www.ubuntu.com/news/UbuntuFoundation

[40] Vgl. O.V., (o.J.),
http://web.archive.org/web/20061201073806/www.sueddeutsche.de/kultur/artikel/826/54772/

[41] Vgl. DistroWatch.com (2004), http://distrowatch.com/dwres.php?resource=popularity

Kriterium	Vorhanden	Kommentar
Namentliche Würdigung von Leistungsträgern	Nicht erkennbar	- Allerdings werden Nachweise über erbrachte Leistungen benötigt, um in der Entwicklungslaufbahn voranzukommen
Events / Messen / Thementage	ja	- regelmäßig Trennung zwischen "Events" + Developer Summit teaminterne IRC-Meetings
Honorar	ja	- als Canonical core developer wird Honorar bezahlt werden, Höhe unbekannt
Wettbewerbe	ja	- Fündig geworden für Artwork - für Devs nicht bekannt, zB. Wallpaper Contest
Entwicklungslaufbahn	ja	- Application Process vorhanden erst "basic" join, dann weiter per "Application" + "Review". - Arbeitsproben für Entwicklung notwendig - sehr gut dokumentierter Prozess, Stufen klar ersichtlich
Grad der Nutzung der Kommunikationskanälen (Twitter, UserGroups)	Ja	- sehr reichhaltig, auch in sozialen Netzwerken vertreten: Youtube, Facebook, Twitter, MailingLists, Forums, UserGroups, Wiki, IRC, Blogs
Feature of the month	Nicht erkennbar	
Developer of the month	Ja	- nicht feature, sondern Featured team

Grundsätzlich kann die Aussage getroffen werden, dass innerhalb der Ubuntu-Community eine Vielzahl von Anerkennungssystemen existieren: So werden beispielsweise Mitglieder öffentlich genannt und vorgestellt, bei Erbringung einer besonderen Leistung erfährt das komplette Team Anerkennung. Desweiteren finden Messen bzw. Events statt, bei denen einerseits die Möglichkeit zu persönlichen Treffen und Gesprächen, andererseits aber auch die Möglichkeit eines Vortrags vor einem größeren Plenum gegeben ist.[42] Gewinner von Wettbewerben (z.B. Design-Wettbewerbe) werden ebenso ausgezeichnet.

Darüber hinause sorgt eine Vielzahl an Informations- und Kommunikationskanälen für einen regen Austausch der Interessierten und Involvierten. Auch wenn dies nicht im eigentlichen Sinne als Anerkennungssystem zu werten ist, steigert man durch eine rege Teilnahme durchaus seinen Bekanntheitsgrad innerhalb der Gemeinschaft. Die Entwicklungslaufbahn mit

[42] Vgl. o.V. (o.J.), http://uds.ubuntu.com/

Zugangsbeschränkungen für „höhere“ Stufen kann auch als reizvoller Faktor gesehen werden, der einen Entwickler länger beschäftigt, weil durch solch ein System die Eigenmotivation der Person geprüft und gefördert wird. Gleichzeitig ist ersichtlich, welches Mitglied zu welchem Grad im Projekt mitarbeitet – ein weiterer Faktor, der Mitgliedern einen „Status“ zuspricht und sie dadurch von anderen abhebt. Ein weiterer Anerkennungsfaktor, der auch mit der Entwicklungslaufbahn zusammenhängt, ist die Möglichkeit auf eine Festanstellung und damit auf ein Honorar. Hierdurch wird kontinuierlich die Arbeit gewürdigt und für viele Projektmitarbeiter ist eine solche Festanstellung eine finale Form der Anerkennung für die eigene Leistung.

Wichtig zu erwähnen sind noch der im Vergleich zu anderen Projekten leichte Einstieg bei Ubuntu: Nicht nur ist im Code of Conduct das wünschenswerte Verhalten der Community-Mitglieder untereinander definiert, es existieren auch zahlreiche Hilfestellungen für Engagement (bspw. beim Aufbau eines lokalen Teams). Ein weiterer Faktor ist das Mentoring-Programm, welches jedem Neuling im Entwickler-Team einen erfahrenen Entwickler (Mentor) für Ratschläge und Hilfe zur Seite stellt. Über diesen Mentor kann auch das persönliche Netzwerk ausgebaut und somit die eigene Leistung über weitere Stellen transparent gemacht werden.

1.3 CMS

Jahia

Jahia reiht sich ein in die Reihe der Open Source Content Management Systeme, die modular aufgebaut sind, dass heißt es bietet einen stabilen Kern an Grundfunktionalitäten. Dieser kann durch Module um weitere Funktionen ergänzt werden. Diese Erweiterungen können dabei von einzelnen Teams unabhängig programmiert werden. Trotz geringer Trefferrate bei dem Google Blog Search Kriterium bietet dieses Projekt eine lange Laufzeit (es wurde 2002 in der Schweiz gegründet) und eine bestehende Community. Jahia hat ein besonderes Geschäftsmodell und ist, trotz der Open Source-Zugehörigkeit, kommerziell ausgerichtet. So ist die Software quellcodeoffen und das Gleiche für alle Benutzer, für Geld jedoch können zusätzliche Leistungen, wie Training, maßgeschneiderte Lösungen oder Module, erkauft werden.[43] Im

[43] Vgl. Jahia Solutions Group SA (2002), http://www.jahia.com/cms/home/about-us/business-model.html

Folgenden wird analysiert, welche Faktoren in diesem Projekt zum Tragen kommen, um die Entwickler nachhaltig anzuerkennen und damit zu motivieren.

Kriterium	**Vorhanden**	**Kommentar**
Namentliche Würdigung von Leistungsträgern	Ja	- Neues aus dem Team / Wochenrückblick
Events / Messen / Thementage	Ja	
Honorar	Ja	- Durch das Konzept von Jahia werden auf der offiziellen Seite Web-Designer und Programmierer zur Implementierung des OSS-Projektes und Beratung für Geld gesucht (http://www.jahia.com/cms/home/about-us/jobs.html).
Wettbewerbe	Nein	- Keine Wettbewerbe auffindbar
Entwicklungslaufbahn	Ja	- Organisation des Projektes in Jira (http://jira.jahia.org/secure/Dashboard.jspa). Wenig persönliche Informationen
Grad der Nutzung der Kommunikationskanälen (Twitter, UserGroups)	Ja	- Foren, Chats, Repository-Plattform: GitHub (https://github.com/jahia)
Feature of the month	Nein	
Developer of the month	Nein	
Sponsoring		Entwickler können sich auf die Jobs des Projektes bewerben und werden angelernt.
Mentoring		Offizielle Unterstützung von Modul-Subgruppen, d.h. Gruppen, die sich mit der Entwicklung von diversen Erweiterungen beschäftigen, hier JahiAppsForge genannt(http://www.jahia.com/cms/home/community/jahiapps-forge.html). Der Fokus liegt hier jedoch mehr auf die Entwicklung von Erweiterungen, während an dem eigentlichen Framework nur ein bezahltes Kernteam arbeitet.

Subgruppenvielfalt	Offizielle Unterstützung von Modul-Subgruppen, d.h. Gruppen, die sich mit der Entwicklung von diversen Erweiterungen beschäftigen, hier JahiAppsForge genannt(http://www.jahia.com/cms/home/community/jahiapps-forge.html). Der Fokus liegt hier jedoch mehr auf die Entwicklung von Erweiterungen, während an dem eigentlichen Framework nur ein bezahltes Kernteam arbeitet.

Jahias Geschäfts-Modell wirkt sich auch auf den Umgang mit der Community aus. Das eigentliche Framework wird von einem bezahlten Kernteam gepflegt und erweitert, die Erweiterungen und Module sind meist Community-gestützt. In der Hierarchie kann eigentlich nur über die angebotenen Jobs aufgestiegen werden. Das diese zahlreich vorhanden sind, weist auf ein funktionierendes Bezahlungskonzept des Projekts hin. Jahia scheint eine kommerzielle Ausrichtung geglückt zu sein, doch geschieht dies mit Abstrichen im Bereich der Community-Größe und der Ausprägung von Sub-Communities (was sich auch in der Google Blog Trefferanzahl zeigt). Eine persönliche Vorstellung einzelner Community-Mitglieder ist nicht aufzufinden. Auch ist eine Subgruppenvielfalt für Module und Erweiterungen vorhanden und wird unterstützt, was für weitere Anerkennung von aktiven Mitgliedern sorgt.

Xoops

Xoops steht für eXtensible Object Oriented Portal System und ist ein Content Management System. Es ist Modular aufgebaut, dass heisst es bietet einen stabilen Kern an, der durch Module um weitere Funktionen ergänzt werden kann. Diese Module können dabei von einzelnen unabhängigen Teams programmiert werden. Das Projekt wurde mehrfach ausgezeichnet, unter anderem auch von dem bekannten deutschen Magazin chip.de[44]. Es unterliegt der GNU General Public License. Mit einer Entwicklungszeit von über 10 Jahren und einer allein auf der Hauptseite 132697 Mitglieder starken Gemeinschaft stellt sich auch in diesem Projekt die Frage, was die aktiven Mitglieder motiviert.

[44] Vgl. Xoops Awards and recognitions (2010), http://www.xoops.org/modules/wfchannel/index.php?cid=28

Kriterium	Vorhanden	Kommentar
Namentliche Würdigung von Leistungsträgern	Ja	User Profile mit diversen Informationen und Aktivität im Forum. Besonders Aktive werden extra mit Bild aufgeführt (http://xoops.org/modules/wfchannel/index.php?pagenum=21).
Events / Messen / Thementage	Nein	
Honorar	Ja	- Es existiert ein Bereich im Forum, in dem Personen Aufträge gegen Bezahlung ausschreiben können. (http://www.xoops.org/modules/newbb/viewtopic.php?topic_id=64294&forum=30)
Wettbewerbe	Nein	
Entwicklungslaufbahn	Ja	
Grad der Nutzung der Kommunikationskanälen (Twitter, UserGroups)	Ja	- Auf Sourceforge vertreten, Mailinglisten, Foren, Chats, Facebook, Flickr, Subgruppen
Feature of the month	Nein	- Nur Rating von Extra-Modulen vorhanden
Developer of the month	Ja	- Es existiert eine Hall of Fame, in der die Entwickler, die besonders zu dem Projekt beigetragen haben, geehrt werden (http://www.xoops.org/modules/wfchannel/index.php?cid=26) und die Innovation Awards, wo Entwickler ausgezeichnet werden, die eine besondere Innovation zu Xoops beigetragen haben. (http://xoops.org/modules/wfchannel/index.php?pagenum=22). Auch ein "Xoopser of the Month" award exisitert, ist aber seit knapp einem Jahr nicht mehr aktuell (stand 01/12) (http://xoops.org/modules/wfchannel/index.php?pagenum=23).

Subgruppenvielfalt	Offizielle Unterstützung von Modul-Subgruppen, d.h. Gruppen, die sich mit der Entwicklung von diversen Erweiterungen beschäftigen. (http://dev.xoops.org/). Nationale Subcommunities, wie die des deutschen Teams (http://www.myxoops.org/), weisen auf eine hohe internationale Verteilung hin.
Mentoring	Zumindest kein offizielles Mentoring, aber durch engen Chat- oder Mail-Kontakt könnte Mentoring trotzdem möglich sein

Die Community von Xoops ist vielfältig ausgerichtet. Sie ist in diversen Untergruppen organisiert und dabei sehr international ausgerichtet.[45] Bemerkenswert sind vor Allem die Förderung von Gruppen, die weitere Module entwickeln wollen und der Bereich im Forum, in dem Unternehmen oder Personen Aufträge gegen ein Honorar im Bereich Xoops verteilen können. Durch die eher geringe Häufung von Personen in einem bestimmten Gebiet existieren keine besonderen Events oder Messen.

Die Existenz von Merchandise weist auf eine hohe Identifizierung der Community mit ihrem Projekt hin, was jedoch nicht direkt einem Anerkennungssystem zuzuordnen ist. Desweiteren ist die hohe Würdigung von Einzelleistungen von Xoops hervorzuheben: Über Awards und einer Hall of Fame werden aktive und tüchtige Mitglieder besonders geehrt.

[45] Vgl. Source Forge (o.J.), http://sourceforge.net/blog/podcast-xoops

Drupal

Drupal gehört ebenfalls zu den sehr häufig genutzten Content Management Systemen.[46] Es steht unter der GNU General Public License. Auch populäre Seiten wie die des englischen Musikfernsehsenders MTV oder die Brainstorm-Website der populären und ebenfalls im Rahmen dieser Arbeit analysierten Linux-Projekt Ubuntu nutzen diese Open Source Technologie.[47] [48] Es ist Modular strukturiert, dass heisst, es besteht zunächst nur aus einem Grundgerüst, das durch Erweiterungen um weitere Funktionen ergänzt werden kann. Diese Module können dabei von einzelnen unabhängigen Teams programmiert werden. Mit mehr als 743,672 registrierten Benutzerkonten allein auf der Drupal-Seite existiert hier auch eine große und vor Allem sehr aktive Gemeinschaft.[49] Ein Blick auf die Motivations- und Organisationstechniken dieses Projektes soll daher weitere wichtige Elemente für diese Arbeit liefern.

Kriterium	Vorhanden	Kommentar
Namentliche Würdigung von Leistungsträgern	Ja	- http://drupal.org/profile, Profil mit diversen Informationen von jedem Mitglied
Events / Messen / Thementage	Ja	- Große Events und Versammlungen mit Sponsoren. Trainings, Messen und sonstige Zusammenkünfte. Äußerst umfangreich und vielfältig.
Honorar	Nein	
Wettbewerbe	Ja	- teilweise auf Veranstaltungen des Projekts
Entwicklungslaufbahn	Ja	
Grad der Nutzung der Kommunikationskanälen (Twitter, UserGroups)	Ja	- IRC (Chat), Blogs, Forum, Mailinglisten und viele weitere durch diverse Subgruppen
Feature of the month	Ja	- Hier werden Module hervorgehoben, die am häufigsten installiert wurden. So findet eine gewisse Anerkennung auch in diesem Bereich statt.

[46] Vgl. Shreves, R. (2010), http://www.waterandstone.com/book/2010-open-source-cms-market-share-report

[47] Vgl. http://www.mtv.co.uk

[48] Vgl. http://drupal.org/node/228222

[49] Vgl. http://drupal.org/getting-involved

Developer of the month	Ja	- Sogenanntes Community Spotlight: Benutzer schlagen andere Benutzer oder Entwickler vor, die es sich ihrer Meinung nach verdient haben. Diese werden dann vorgestellt und geehrt.
Subgruppenvielfalt		http://groups.drupal.org/, offizielle Unterstützung und eine hohe Vielfalt an regionalen und international-orientierten Subgruppen, die sich mit dem OSS-Projekt beschäftigen
Mentoring		ja, teilweise in Subgruppen und Trainings (Events)

Das Open Source Content Management Projekt Drupal setzt sehr auf eine enge und persönliche Kommunikation der Mitglieder. Die gebotenen Kommunikationswege sind vielfältig, jeder Benutzer kriegt ein Konto, welches er mit persönlichen Informationen füllen kann und das Forum ist äußerst aktiv. Durch die hohe Vielfalt an Messen und Events in allen Teilen der Welt kann ebenfalls ein persönlicher Kontakt hergestellt werden. Das sogenannte "Community Spotlight" sorgt für die Vorstellung von immer neuen aktiven Mitgliedern und gibt den Partizipierenden ein Gefühl der besonderen Wertschätzung. In diesem Projekt werden auch die Subgruppen enorm gefördert. Man bietet ein Verzeichnis und fördert explizit auch kommerzielle Consulting-Gruppen, die mit Drupal arbeiten.

Joomla

Joomla ist eines der meistgenutzten CMS weltweit und steht unter der General Public License.[50] Joomla ging aus dem Projekt Mambo nach Streitigkeiten mit einem Unternehmen, welches die Namensrechte an Mambo besaß, hervor. Es ist Modular aufgebaut, dass heisst es bietet einen stabilen Kern an, der durch Module um weitere Funktionen ergänzt werden kann. Diese Module können dabei von einzelnen unabhängigen Teams programmiert werden. Was konnte die Community dieses erfolgreichen OS-Projektes motivieren und wodurch wird Anerkennung an Mitglieder ausgesprochen? Worauf setzt die Joomla Community und was sind die Erfolgsfaktoren, die die Beteiligten motivierten und motivieren? Der folgende Kriterienkatalog liefert Antworten auf diese Fragen.

[50] Vgl. http://www.idealware.org/reports/2010-os-cms?key=45152768, vgl. dazu auch Shreves, R. (2010), http://www.waterandstone.com/book/2010-open-source-cms-market-share-report

Kriterium	Vorhanden	Kommentar
Namentliche Würdigung von Leistungsträgern	Nein	- nicht wichtig, andere Priorisierung im Projekt
Events / Messen / Thementage	Ja	- Joomladays, Joomla User Groups Treffen
Honorar	Ja	- Extensions können als kostenpflichtig bereitgestellt werden
Wettbewerbe	ja	- Wöchentliches Treffen der Entwickler ("Get Together")
Entwicklungslaufbahn	ja	- sechs Wochen Einstiegskurs, Topentwickler werden mit Bild und Lebenslauf auf der Seite vorgestellt
Grad der Nutzung der Kommunikationskanälen (Twitter, UserGroups)	ja	- Reichhaltig, IRC, Wikis, Subcommunities, Forums, Skype, Mailinglisten - Alle Möglichkeiten sind durch zahlreiche inoffizielle Subcommunities garnicht zu zählen
Feature of the month	ja	- Erweiterungen mit Kategorien "new and noteworthy" - Ratingsystem für einzelne Erweiterungen des großen Extensionsystems
Developer of the month	Ja	- nur All-Time Top-Developer
Subgruppenvielfalt		- http://community.joomla.org/user-groups.html, offizielle Unterstützung und eine hohe Vielfalt an regionalen und international-orientierten Subgruppen, die sich mit dem OSS-Projekt beschäftigen
Mentoring		- Ja, in bestimmten Subcommunities

Ein Großteil der Motivation der Joomla-Community kommt durch die Eigendynamik, erreicht durch die breite Benutzerbasis des Werkzeuges und der zahlreichen Subcommunities. Gerade das Vorkommen der vielen großen und kleinen inoffiziellen Subcommunities gantiert dabei eine hohe Dezentralisierung. Dies bietet zahlreiche Vor- und Nachteile: Der Einstieg als Entwickler fällt durch die primäre Unübersichtlichkeit schwieriger aus, ist man aber involviert, garantieren lokale Treffen, nahe Benutzergruppen und die hohe Nachfrage eine hohe Langzeitmotivation und Wertschätzung. Anerkennung wird im Joomla-Projekt dabei besonders durch das persönliche Vorstellen von Top-Entwicklern und das umfangreiche Bewertungssystem der einzelnen Module verliehen.

Typo 3

Auch Typo3 gehört zu den martkanteilsstärksten und wichtigsten CMS-Systemen weltweit. Es steht unter der GPL und wurde ursprünglich von Kasper Skårhøj entwickelt. Die fortgeschrittene Community, die sich auch in vielen lokalen Stammtischen sammelt, ist ein gutes Beispiel für aktive Open Source Kollaboration und soll deswegen näher betrachtet werden. Wie werden die Community und die Entwickler hier anerkannt und motiviert?

Kriterium	Vorhanden	Kommentar
Namentliche Würdigung von Leistungsträgern	Nein	
Events / Messen / Thementage	Ja	- http://association.typo3.org/activities/events/ - - TYPO3 Snowboard Tour, jährliche TYPO3 Konferenzen, T3CON oder T3DD - TYPO3 Entwicklertage, diverse durch die Community gestütze Events
Honorar	Nein	
Wettbewerbe	ja	- eventuell auf Events
Entwicklungslaufbahn	Ja	- http://forge.typo3.org/users
Grad der Nutzung der Kommunikationskanälen (Twitter, UserGroups)	Ja	- Einteilung in verschiedene Teams - Maillisten, Blogs, IRC, Subgruppen
Feature of the month	Nein	
Developer of the month	Nein	
Zertifizierungsmöglichkeiten		
Subgruppenvielfalt		- http://typo3.org/community/typo3-user-groups/, offizielle Unterstützung und eine hohe Vielfalt an regionalen und teilweise international-orientierten Subgruppen, die sich mit dem OSS-Projekt beschäftigen
Mentoring		- Über die UserGroups

Die hier analysierten CMS-Open Source Community nutzt Kommunikationstools wie IRC, Blogs und vor allem Mailinglisten. Daneben sind vor allem die zahlreichen Events und Stammtische hervorzuheben, auch wenn diese nicht direkt einem Anerkennungssystem zuzuordnen sind.
In der Typo3-Community finden dabei viele einzelne Benutzergruppen Anwendung. Organisiert und gruppiert sind diese dabei meist dezentralisiert. Die Struktur ist lose gehalten, sodass sich

Subcommunities frei entfalten können. Regionale Stammtische und Gruppen sorgen für viel persönlichen Kontakt insbesondere in Deutschland. Die Ergebnisse der einzelnen Gruppen werden zentral über Mailinglisten gesammelt und durch das sogenannte "Core Team" umgesetzt. Diese Hierarchie und insbesondere die Möglichkeit in dieser aufzusteigen sind Möglichkeiten, als Entwickler in diesem Projekt Anerkennung zu finden. Änderungen im Code können auch vorgeschlagen werden, ohne dem "Core Team" zugehörig zu sein. Durch die Unterstützung von Unterprojekten und der persönlichen Vorstellung eigener und externer Teams werden akitve Mitgliedern gewürdigt und motiviert.

1.4 Administration

Apache

Die Apache Software Foundation ist eine Non-Profit Organisation registriert in Amerika. Mitglieder aus aller Welt unterstützen seit 1999 die Entwicklung der Organisation, die sich aus über 100 Top-Level Projekten zusammensetzt und besonders durch ihre Web-Server Technologie hervortut. Durch den Non-Profit Charakter sind keine Entwickler angestellt, es werden keine Honorare gezahlt. Die anfallenden Infrastrukturkosten werden durch Spende gedeckt, Sponsoren werden an verschiedenen Orten vor- und dargestellt als Gegenleistung.

Kriterium	Vorhanden	Kommentar
Namentliche Würdigung von Leistungsträgern	ja	Twitterfeeds werden auf der Hauptseite gefeatured (Fixes, Releases, Sonstiges); Mailing Lists heben Engagement und Beitrag der Individuen hervor
Events / Messen / Thementage	Nein	
Honorar	Nein	

Wettbewerbe	Ja	Teilweise Nutzung von Wettbewerben zur Ergebnisförderung: Logo-Design Wettbewerb etc.
Entwicklungslaufbahn	Ja	Meritocracy, User, Developer, Committer, diverse andere Rollen. Je nach Engagement bekommen Teilnehmer mehr Rechte und Verantwortung zugeteilt. Project Management Commitee vorhanden als Führung des Projektes und einzelner Teilprojekte, gewählt von der Community.
Grad der Nutzung der Kommunikationskanälen (Twitter, UserGroups)	Ja	Twitter, Blogs, Mailing Lists, IRC
Feature of the month	Nein	
Developer of the month	nein	

Unter dem Aspekt der Anerkennungssysteme ist die ASF interessant zu betrachten. Dies rührt aus zwei auf den ersten Blick konträren Fakten: Zum einen gibt es keinerlei monetäre Vergütung für Entwickler, jegliche Arbeit wird nur durch Respekt und Anerkennung honoriert. Zum anderen ist das Projekt sehr erfolgreich und seit langer Zeit beständig, obwohl kaum öffentliche Anerkennungssysteme erkennbar sind. Öffentlich bedeutet, dass Mitglieder sichtbar werden zu Nicht-Mitgliedern (z.b. Personen, die nicht auf der Mailing List stehen). Andererseits werden alle Twitter-Aktivitäten auf der Apache Hauptseite angezeigt mit Namen der Contributer. Auch das Fehlen von Zentral organisierten Kongressen und / oder Thementagen ist bemerkenswert. Offentsichtlich geht es bei diesem Projekt hauptsächlich um die Anerkennung anderer Spezialisten und Fachleuten, nicht um die Anerkennung und Sichtbarwerdung der Öffentlichkeit gegenüber.

JBoss

JBoss ist ein zu Red Hat zählendes Unternehmen, welches sich auf die Entwicklung rund um den JBoss Application Servers spezialisiert hat. Das Unternehmen agiert nach dem selbst aufgestellten Modell des "Professional Open Source", wobei im Open Source Stil Software

entwickelt aber kommerziell vertrieben und unterstützt wird. Hierbei werden speziell eingestellte Entwickler beschäftigt, wie in einem traditionellen Unternehmen.

Kriterium	Vorhanden	Kommentar
Namentliche Würdigung von Leistungsträgern	ja	Member unterteilt in Developer, Writer etc., werden in der Community mit Bild gezeigt; Top-Teilnehmer werden herausgestellt; Kürzlich beigetretene Mitglieder werden besonders hervorgehoben
Events / Messen / Thementage	Ja	Treffen selbstorganisiert in diversen Großstädten global
Honorar	Ja	"Professional Open Source", getrieben und finanziert von Red Hat
Wettbewerbe	Nein	
Entwicklungslaufbahn	Ja	Red Hat Certifications möglich; JBoss Certifications möglich; bei mehr und höherer Zertifizierung wird mehr Verantwortung übertragen
Grad der Nutzung der Kommunikationskanälen (Twitter, UserGroups)	Ja	User Groups, LinkedIn, Foren werden genutzt
Feature of the month	Nein	
Developer of the month	Ja	"JBoss Community Recognition Award", einmal jährlich werden besonders engagierte Individuen von ihren Kollegen nominiert und ausgezeichnet

Red Hat scheint ein treibender Faktor hinter diesem Projekt zu sein. JBoss sichert sich das Engagement seiner Entwickler, indem es sie einstellt und ihnen eine bezahlte Arbeitsstelle bietet. Die öffentlich sichtbare Community-Beteiligung beschränkt sich auf Forenbeiträge und User Groups, die Beteiligung wirkt zurückhaltend, verhalten. Herauszustellen ist der jährlich verliehene Community Award und die stattfindenden Treffen der Mitglieder. Die Struktur von JBoss wirkt eher konservativ durch die klassisch beschäftigten Key-Contributer und die sichtbaren Anerkennungssysteme lassen sich kaum nach ihrer Effiziens beurteilen, da der vordergründige Motivationsfaktor im Projekt das erhaltene Honorar zu sein scheint.

MySQL

MySQL ist ein sich mittlerweile unter der Kontrolle von Oracle befindliches relationales Datenbanksystem. Oracle hat angestellte Entwickler für die proprietären Zweige von MySQL und stützt sich bei dem reinen Open Source Teil von MySQL auf die Entwicklercommunity.

Kriterium	Vorhanden	Kommentar
Namentliche Würdigung von Leistungsträgern	ja	Bei Codesnippets und Dokumenten werden credits gegeben, allerdings nicht speziell gefeatured
Events / Messen / Thementage	Ja	
Honorar	Ja	Oracle führt mySQL fort nach dem Kauf 2008 mit angestellten Entwicklern
Wettbewerbe	Nein	
Entwicklungslaufbahn	Ja	"mySQL Quality Contribution Program" beinhaltet eine Laufbahn für aktive Mitglieder der Community. Je nach Anzahl der Bugfixes, Committs etc. erhalten diese höhere Ränge und bekommen werden sichtbarer zu Entscheidungsträgern und Mitgliedern in höheren Positionen. Zudem dürfen sie an entscheidenderen Projekten mitarbeiten, wie den Platinum Enterprise Editions
Grad der Nutzung der Kommunikationskanälen (Twitter, UserGroups)	Ja	Mailinglisten, Foren
Feature of the month	Nein	
Developer of the month	Nein	
Top ranked XY	Ja	Herausstechende Projekte, Codesnippets etc. werden auf der mySQL Startseite gefeatured

Die hohe Popularität und die starke Unterstützung von Oracle sind Gründe für den Erfolg von MySQL. Anerkennungssysteme sind teilweise vorhanden. Es gibt eine in mehreren Dimensionen stattfindende Entwicklerlaufbahn und als "Top" eingeordnete Dinge von Interesse (Projekte, Code etc.) werden gefeatured. Die Community ist zudem sehr aktiv und es gibt

mehrere Kongresse und Blogs, die sich mit MySQL befassen. Dies sind günstige Gelegenheiten für Entwickler sichtbar zu werden und ihre Qualitäten herauszustellen. Die starke Unterstützung durch Oracle scheint der Motivation der Entwickler keinen Abbruch zu tun, das Interesse an MySQL Community-seitens ist weiterhin ungebrochen.

Palo Suite

Die von Jedox unter einer Dualen Lizenz vertrieben Palo Suite ist ein Business Intelligence Gesamtpaket. Es bietet einen OLAP Server, Excel Plugins für gemeinsames Arbeiten, eine Web Oberfläche sowie einen ETL Server zum Importieren von externen Datenquellen wie Data Warehouses. Jedox bietet um ihre Palo Suite diverse Dienstleistungen und Zusatzangebote, dies beinhaltet auch Schulungen und Seminare. Die BI Lösung ist bei Unternehmen beliebt und Jedox befindet sich mit der Umwandlung zu einer Aktiengesellschaft 2008 in einem wirtschaftlichen Wachstum.

Kriterium	**Vorhanden**	**Kommentar**
Namentliche Würdigung von Leistungsträgern	ja	Im Source Forge Repository beim Durchführen von Änderungen.
Events / Messen / Thementage	Ja	Jährlich stattfindender Kongress, Teilnahme auf anderen OS Kongressen
Honorar	Ja	Vermutlich Community Manager
Wettbewerbe	Nein	
Entwicklungslaufbahn	Nein	
Grad der Nutzung der Kommunikationskanälen (Twitter, UserGroups)	Ja	Foren, Source Forge Repository
Feature of the month	Nein	
Developer of the month	Nein	

Die von Jedox getriebene Palo Suite weist kaum Anerkennungssysteme auf. Öffentlich erkennbar sind klassische Kommunikationsplattformen wie Foren, Boards und Unternehmensgesteuerte Treffen. Die Veranstaltung eines Hauseigenen Kongresses scheint die Entwickler zusammenzubringen und zu motivieren. Die Teilnahme an anderen Messen und

Kongressen scheint Jedox und dem Palo Suite Projekt ebenfalls von nicht zu unterschätzender Bedeutung zu sein. Es fällt auf, dass keine Einzelpersonen - öffentlich - hervorgehoben oder besonders gelobt werden o.Ä. Im Gegenteil, wenn Entwicklern ihre Arbeit angerechnet wird, dann im Zuge eines sowieso stattfindenden Personentrackings im Source Forge Repository. Obwohl kaum Anerkennungyssysteme erkennbar sind scheint die Palo Suite sehr beliebt zu sein, zudem weisen die Community-Foren bis zum heutigen Tag rege Beteiligung auf.

Xen

Xen ist ein Hypervisor, eine Software zur Überwachung und Steuerung von verschiedenen Virtuellen Maschinen auf einem phyischen Computer. Es entwickelt sich derzeit zu einem De Facto Standard und wird von vielen, mitunter konkurrierenden Firmen wie IBM, HP, Oracle und Microsoft unterstützt. Xen wird unter der GNU GPL v2 vertrieben.

Kriterium	Vorhanden	Kommentar
Namentliche Würdigung von Leistungsträgern	ja	Mitglieder werden mit Bildern vorgestellt
Events / Messen / Thementage	Ja	Xen Summits, Xen Days, explizit genannt und veranstaltet
Honorar	Nein	
Wettbewerbe	Nein	
Entwicklungslaufbahn	Ja	Meritocracy, je mehr beigetragen wird, desto mehr Verantwortung wird übertragen; Verschiedene Rollen wie Maintainer, Committer, Project Lead, Mentor etc. vorhanden; genaue Definitionen und Mechanismen für die Rollen vorhanden; Möglichkeit einer Festanstellung der sich beteiligenden Firmen besteht bei Mitarbeit
Grad der Nutzung der Kommunikationskanälen (Twitter, UserGroups)	Ja	Mailing Lists, IRC, Blog, Twitter, User Groups, diverse Soziale Medien
Feature of the month	Nein	
Developer of the month	Nein	
Papers	Ja	Aus der Community entstandene Paper werden mit credits vorgestellt

Auffällig bei Xen sind die ordentlich geplanten und organisierten Kongresse und Entwicklertreffen. Außerdem existiert eine klar aufgestellte Entwicklerlaufbahn in die die Mitglieder hineinwachsen können. Interessant ist zudem die Möglichkeit für aktive Beitragleistende Stellenangebote von sponsorenden Firmen zu erhalten. Für die Öffentlichkeit erkennbare Anerkennungssysteme sind kaum vorhanden, der Einblick in die internen Mail Listen ist leider nicht möglich an dieser Stelle.

1.5 Büroanwendungen und Multimedia

Tulip

Tulip ist ein Framework, das Daten analysiert und visualisiert. Die Lizenz ist die LGPL.

Kriterium	Vorhanden	Kommentar
Namentliche Würdigung von Leistungsträgern	Ja	- Neues aus dem Team / Wochenrückblick
Events / Messen / Thementage	Nein	
Honorar	Nein	
Wettbewerbe	Nein	
Entwicklungslaufbahn	Nicht erkennbar	
Grad der Nutzung der Kommunikationskanälen (Twitter, UserGroups)	Ja	- gering
Feature of the month		
Developer of the month		

Als Anerkennung für die Entwickler werden sie namentlich auf der Website des Projektes erwähnt. Ansonsten gibt die Community gibt wenig Informationen über sich Preis.

LibreOffice

LibreOffice is ein Officepaket und 2010 aus dem Open Software Projekt entstanden um ein von Oracle unabhängiges Projekt zu gewährleisten. LibreOffice bietet ein freies Office Packet mit verschiedenen Modulen, die denen von MS Office ähneln, an. Es wird jedoch unter der LGPL Lizenz veröffentlicht. Getragen wird das Projekt von einer Stiftung.

Kriterium	Vorhanden	Kommentar
Namentliche Würdigung von Leistungsträgern	Ja	- Namentliche Nennung der Mitglieder
Events / Messen / Thementage	Ja	- Zusammentreffen als Motivation und Austauschmöglichkeit
Honorar	Nein	- ehrenamtliche Tätigkeit
Wettbewerbe	Nicht erkennbar	
Entwicklungslaufbahn	Ja	- keine Einstiegsbarriere; verschiedene Aufgaben je nach Erfahrung, verscheidenene Aufgaben auch ohne Programmierung
Grad der Nutzung der Kommunikationskanälen (Twitter, UserGroups)	Ja	- gering: Mail und Chat
Feature of the month		
Developer of the month		

Die Mitglieder dieser Community erhalten Anerkennung durch die Nennung als Redner bei Veranstaltungen und die Möglichkeit sich innerhalb der Community hochzuarbeiten. Dabei gibt es verschiedene Aufgaben je nach Erfahrung der Mitglieder. Während Events oder Veranstaltungen können Mitglieder reden und werden später dann auf der Website der Community genannt. Desweiteren werden alle Mitglieder, die seit Gründung Codezeilen eingeschickt haben, namentliche erwähnt.

kMeleon

K-Meleon ist ein Webbrowser, der besonders schlank und schnell ist. Er basiert auf der Layout Engine, die Mozilla entwickelt hat und auch für den Firefox verwendet. Ein einschränkendes Merkmal ist, das der Browser speziell für Windows entworfen wurde. Es steht seit Beginn unter der GPL Lizenz und wird nur von einer kleinen Community weiterentwickelt.

Kriterium	Vorhanden	Kommentar
Namentliche Würdigung von Leistungsträgern	Nein	
Events / Messen / Thementage	Nicht erkennbar	
Honorar		
Wettbewerbe		
Entwicklungslaufbahn	Ja	- verschiedene Rollen innerhalb des Teams
Grad der Nutzung der Kommunikationskanälen (Twitter, UserGroups)	Nicht erkennbar	
Feature of the month		
Developer of the month		

Die Community basiert auf einer Mailingliste und den Einträgen bei Sourceforge. Ansonsten gibt das Projekt keine Informationen nach außen preis.

Firefox

Firefox ist ein Teil des Mozilla Projektes. Er ist ein freier Webbrowser. Die Besonderheit des Browsers besteht in den vielen Anpassungsmöglichkeiten wie Designvorlagen und Add-ons. Schirmherr ist die Moziall Foundation, eine Non-Profit-Organisation, die die rechltlichen und infrastrukturellen Grundlagen bietet. Der Webbrowser selbst wurde 2002 noch unter dem namen Phoenix 0.1 als ein Bestandteil von Mozilla veröffentlicht. Lizenziert wird die Software durch verschiedene Lizenzen (MPL/GPL/LGPL).

Kriterium	Vorhanden	Kommentar
Namentliche Würdigung von Leistungsträgern	Ja	- im Profil werden die Bewertungen der Add-Ons dargestellts
Events / Messen / Thementage	Ja	- Stammtisch - Konferenzen - Download days
Honorar	Nein	
Wettbewerbe	Ja	- Designwettbewerbe
Entwicklungslaufbahn	Ja	- verschiedene Rollen innerhalb des Teams
Grad der Nutzung der Kommunikationskanälen (Twitter, UserGroups)	Ja	- Forum (Mozilla Projekt), Twitter account, News, Google Groups, Newsgroups
Feature of the month	Ja	- es werden verschiedene Add-ons vorgestellt
Developer of the month	Nicht erkennbar	

Mitglieder in dieser Community haben die Möglichkeit durch Add-ons an der Anpassbarkeit des Browsers mitzuarbeiten. Diese Add-ons stellen häufig Subprojekte da, an denen mehrere Entwickler mitarbeiten. Die besten und beliebesten Add-ons werden auf der Website des Projektes präsentiert. Daneben haben die Anwender die Möglichkeit sich die Entwickler eines Add-ons näher ansehen. Dafür kann sich jeder Entwickler auf einer eigenen Website präsentieren. Daneben besteht für die Entwickler die Möglichkeit über gute Entwicklungen sich für einen Job bei der Mozilla Foundation zu empfehlen. Dafür ist es auch von Bedeutung, dass es verschiedene Aufstiegsmöglichkeiten gibt.

7zip

7-Zip ist ein Datenkompressionsprogramm und wurde 1999 von Igor Wiktorowitsch Pawlow veröffentlicht. Die Software steht unter der LGPL Lizenz.

Kriterium	Vorhanden	Kommentar
Namentliche Würdigung von Leistungsträgern	Nicht erkennbar	
Events / Messen / Thementage	Nein	
Honorar	Nein	
Wettbewerbe	Nicht erkennbar	
Entwicklungslaufbahn	Nicht erkennbar	- Anfrage an Admin um Teilzunehmen ansonsten wenige Infos
Grad der Nutzung der Kommunikationskanälen (Twitter, UserGroups)	Nicht erkennbar	
Feature of the month	Nicht erkennbar	
Developer of the month	Nicht erkennbar	

Diese Community macht kaum Angaben. Es wird lediglich Igor Wiktorowitsch Pawlow als Administrator erwähnt. Es wird kein Entwickler hervorgehoben und es gibt keine Informationen über die genaue Zusammenarbeit innerhalb der Community.

1.6 Softwareentwicklung

Eclipse

Eclipse ist eine Software-Entwicklungsumgebung die im Jahr 2001 durch die IBM, welche das Produkt ursprünglich als proprietäre Anwendung gedacht hatte, zu einer OSS Anwendung überführt wurde. Die administrative Steuerung des Projekts erfolgt gegenwärtig durch die Eclipse Foundation. Diese ist eine gemeinnützige Organisation, die z.B. Aufgaben wie die Verwaltung von Spendengeldern oder den Dialog mit kommerziell orientierten Unternehmen ausübt. Auf Grund der Größe des Projekts befinden sich innerhalb der Eclipse Foundation Arbeitskräfte, die gegen Entgelt Vollzeit beschäftigt sind. Diese wirken laut Internetauftritt bei genannten übergreifen Aufgaben mit.

Kriterium	Vorhanden	Kommentar
Namentliche Würdigung von Leistungsträgern	Ja	- Sehr ausgeprägt: Hall of fame, Weltkarte Emeritus Alumni, Project maturity system
Events / Messen / Thementage	Ja	- z.B. eclipsecon, eclipse days - Auftritt von Eclipse Foundation Vertretern als Stargäste - Betreung von User Groups
Honorar	Ja	-Subvention von Tagungen, Infrastrukturkosten der einzelnen Subprojekte durch die Eclipse Foundation - keine direkte Zahlung an Entwickler
Wettbewerbe	Ja	- Monatliche Programmier-/ Designwettbewerbe
Entwicklungslaufbahn	Ja	- Komplexer und streng regulierter Prozess (öffentlich und transparent) - board of directors wird gewählt -Heroisierung
Grad der Nutzung der Kommunikationskanälen (Twitter, UserGroups)	Ja	-hochfrequente Nutzung einer Großzahl gängiger Plattformen und Kanäle allerdings meist anonym
Feature of the month	Nein	
Developer of the month	Nein	

Wie in der Tabelle oberhalb zu sehen, zeichnet sich das Eclipse Projekt durch eine ausgeprägte Nennung von Leistungsträgern gegenüber seinem Umfeld aus. Aktuelle und ehemalige Mitarbeiter werden explizit genannt, beispielsweise auf einer Weltkarte die darstellt an welcher geografischen Position diese wohnhaft sind. Ferner existiert einer Art von „Hall of fame“ in der eine kleine Anzahl von Personen in besonderem Maße für ihren Beitrag zum Projekt gewürdigt wird. Ebenso geschieht dies für die Gewinner der monatlichen Programmier- und Design-Wettbewerbe.

Hinsichtlich von Tagungen und Treffen weist das Projekt ebenfalls eine vergleichsweise Hohe Aktivität vor. Es gibt mehrere internationale und mehrtätige Veranstaltungen, auf denen Repräsentanten des Projekts als auch Vertretern der Sponsoren des Projekts vor Publikum sprechen. Diese Veranstaltungen werden durch die Eclipse Foundation finanziert, welche das Kapital des Projekts verwaltet. Zwar existieren keine direkten Honorarzahlungen, dennoch gibt es über die Finanzierung von eben genannten Veranstaltungen als auch sonstiger Infrastruktur, einen indirekten finanziellen Ausgleich für die Aufwendungen, die die Mitarbeit an Eclipse bedeutet.

Ähnlich zur Personalorganisation in kommerziell ausgerichteten Wirtschaftsunternehmen, weist Eclipse eine hierarische streng regulierte Mitarbeiterentwicklung auf. Zugrundeliegende Prozesse sind öffentlich und werden transparent beschrieben. Als Anerkennung für qualitativ hochwertige Leistungen, wird einer Person nach und nach mehr Macht innerhalb des Projekts eingeräumt und aufgetragen. Diese steigt in der internen Hierarchie auf.

Mit Blick auf die Nutzung gängiger Kommunikationskanäle kann Eclipse ein hoher Grad an Aktivität zugesprochen werden. Das Projekt ist auf einer Vielzahl von Kanälen hochfrequent vertreten. Allerdings muss festgestellt werden, dass zugehörige Beiträge im Gegensatz zu anderen Projekten meist anonym verfasst werden.

jQuery

Die ECMA-Script (Java Script) Bibliothek jQuery, die im Jahr 2006 erstmalig durch John Resig veröffentlicht wurde, dient zur Vereinfachung der Entwicklung von Webseiten. Sie erleichtert u.a. Aufgaben wie Benutzerinterkation oder AJAX-Datenzugriffe. Aus organisatorischer Sicht ist jQuery dreigeteilt. Die Führungsspitze bildet das jQuery Board. Darunter findet sich das sogenannte Team, zu welcher nach einem Auswahlprozess Personen aus der Menge an Contributern stoßen können. jQuery ist gegenwärtig sehr populär und nach t3n eines der meistgefragten Open Source Projekte.

Kriterium	Vorhanden	Kommentar
Namentliche Würdigung von Leistungsträgern	Ja	- Namentliche Nennung aller Mitglieder inklusive Rolle und Position - Auflistung ehemaliger Mitglieder - Featured members der jQuery meetup groups auf Weibte
Events / Messen / Thementage	Ja	- jQuery Conferences, Auftritt der Team Member als Stargäste
Honorar	Ja	- Beantragung von Aufwandsentschädigung
Wettbewerbe	Nein	
Entwicklungslaufbahn	Ja	-Contributer,Team Member,Board Member -Heroisierung
Grad der Nutzung der Kommunikationskanälen (Twitter, UserGroups)	Ja	- Keine überdurchschnittlich starke Nutzung - Blog auf homepage
Feature of the month	Nein	
Developer of the month	Nein	

Hinsichtlich der Nennung von Leistungsträgern lässt bei jQuery feststellen, dass alle gegenwärtigen festen Mitglieder namentlich auf der Webseite des Projekts genannt werden. Zudem werden ausgewählte Personen aus den jQuery meetup groups besonders hervorgehoben. Auf der Hausmesse jQuery Conference treten Projektmitglieder ferner in hervorgehobener Rolle als Referenten auf. Generell existiert eine klare organisatorische Hierachie.

Mit Blick auf finanzielle Anerkennung lässt sich sagen, dass das Projekt die Möglichkeit anbietet, finanzielle Aufwandsentschädigung für erbrachte Leistungen zu beantragen. Die Wahrscheinlichkeit der Genehmigung ist jedoch nicht abzuschätzen.

Ruby on Rails

Das Web Anwendungs Framework Ruby on Rails (RoR) wurde von David Heinemeier für die Programmiersprache Ruby entwickelt und im Jahr 2004 initial veröffentlicht. Es folgt dem MVC-Prinzip und eignet sich auf Grund dem strikt gelebten Paradigma „Don't repeat yourself“ (DRY) sowie der Integration weiterer Frameworks zur Umsetzung agiler Entwicklungsprinzipien. Das Projekt wird durch das sogenannte Core Team gesteuert und nach außen repräsentiert.

Kriterium	Vorhanden	Kommentar
Namentliche Würdigung von Leistungsträgern	Ja	-Core Team: Bilder, kurze Vorstellung jedes Mitglieds, Nennung ehemaliger Mitglieder - Auflistung der Contributer inklusive der Nennung der Anzahl von Commits
Events / Messen / Thementage	Ja	- Hausmesse: Rails Conf - User Groups
Honorar	Nein	
Wettbewerbe	Ja	- z.B. Rails Rumble
Entwicklungslaufbahn	Ja	- Contributer, Core Team - Listung des Core Teams absteigend nach Grad der internen Reputation - Heroisierung
Grad der Nutzung der Kommunikationskanälen (Twitter, UserGroups)	Ja	- Core Team wirbt mit echten Namen auf Twitter - Blog auf Homepage - Serien von Webcasts der Core Team Mitglieder
Feature of the month	Nein	
Developer of the month	Nein	

Bei RoR werden alle zum Projekt zählenden Personen namentlich und mit der Anzahl der Commits im Sourcecode Repository genannt. Die Mitglieder des Core Teams, welches das Projekt leitet, werden in besonderem Maße hervorgehoben. Jedes dieser Mitglieder wird mit Foto und Kurzprofil vorgestellt. Zudem werden die Mitglieder des Core Teams absteigend nach Grad der internen Reputation gelistet. Hierbei ist eine Art von Heroisierung dieser hochrangigen Mitglieder festzustellen.

Gängige Kommunikationskanäle wie Twitter oder Facebook werden durch RoR kontinuierlich bedient. Hierbei werden die Beiträge im Namen einzelne Mitglieder verfasst. Zum Teil mit

Bildern von diesen. Auf der Webseite des Projekts ist zudem ein Blog sowie eine Serie von Webcasts der Core Team Mitglieder zu finden.

Notepad++

Notepad++ ist ein umfangreicher Texteditor der durch den Franzosen Don Ho entwickelt wurde. Neben ihm gehört zum Projekt eine Reihe von Personen, die als Code Contributer geführt werden. Gegenwärtig ist es rein für Windows Systeme verfügbar.

Kriterium	Vorhanden	Kommentar
Namentliche Würdigung von Leistungsträgern	Ja	- Namentlich gelistet (eher klar unter dem Erfinder untergeordnet) - Entwickler sind in der Anwendung hinterlegt
Events / Messen / Thementage	Nein	
Honorar	Nein	
Wettbewerbe	Nein	
Entwicklungslaufbahn	Nein	
Grad der Nutzung der Kommunikationskanälen (Twitter, UserGroups)	Ja	- Aktive Nutzung von Facebook + Twitter
Feature of the month	Nein	
Developer of the month	Nein	

Alle Personen die einen Beitrag zum Projekt geliefert haben werden auf dessen Webseite als auch in der Anwendung selber aufgelistet. Hierbei ist jedoch stets eine Distanzierung zum Gründer Don Ho zu beobachten. Hinweise auf Veranstaltungen wie Programmierwettbewerbe oder Messen sind nicht zu finden. Diese repräsentiert das Projekt nach außen u.a. durch Nutzung von Facebook und Twitter.

FOXOpen

Durch die britische Behörde „Departement of Energy and Climate Change“ (DECC) entwickelt und erst nachträglich als Open Source Software veröffentlicht, ist FOXOpen ein Framework, dass der Entwicklung und Bereitstellung von E-Business Anwendungen dient. Es dient u.a. der Abbildung von administrativen Prozessen samt zugehörigen Eingabeformularen. Gegenwärtig wird FOXOpen durch die DECC jedoch lediglich nur noch bezuschusst.

Kriterium	Vorhanden	Kommentar
Namentliche Würdigung von Leistungsträgern	Nein	-Anwendung wurde von einer britischen Behörde entwickelt und erst nachträglich zu OSS umgewandelt. Es ist keine namentliche Nennung jeglicher Entwickler zu finden
Events / Messen / Thementage	Nein	
Honorar	Nein	
Wettbewerbe	Nein	
Entwicklungslaufbahn	Nein	-Es gibt keine Indizien für die Möglichkeit einer Beteiligung am Projekt
Grad der Nutzung der Kommunikationskanälen (Twitter, UserGroups)	Nein	
Feature of the month	Nein	
Developer of the month	Nein	

Wie in der Tabelle dargestellt, existiert kein Indiz auf eine gegenwärtige Weiterentwicklung des Projekts als auch der Möglichkeit der Teilnahme.

EiffelStudio

EiffelStudio ist die Entwicklungsumgebung zur Programmiersprache Eiffel, welche 1985 vom französischen Informatiker Bertrand Meyer entwickelt wurde. Das Produkt gehört dem Unternehmen Eiffel Studio welches dessen Administration als auch Bereitstellung leitet.

Kriterium	Vorhanden	Kommentar
Namentliche Würdigung von Leistungsträgern	Ja	- Namentliche Nennung der Zugehörigkeit zu einem der Entwicklungsteams -Nach außen präsentiert sich Eiffel primär noch wie ein kommerzielles Produkt, Entwicklernamen sind nur schwer zu finden
Events / Messen / Thementage	Nein	
Honorar	Ja	-für richtige Jobs: ja -für ehrenamtliche Mitarbeiter: keine Angabe
Wettbewerbe	Nein	
Entwicklungslaufbahn	Ja	'-1) Patches/BugFixes 2) Schreibzugriff auf den Programmcode
Grad der Nutzung der Kommunikationskanälen (Twitter, UserGroups)	Nein	
Feature of the month	Nein	
Developer of the month	Nein	

Die öffentliche Würdigung von Leistungsträgern erfolgt an versteckter Stelle. Innerhalb der Wissensdatenbank des Projekts existiert eine Zuordnung von Personen zu Entwicklungsteams. Primär jedoch präsentiert sich EiffelStudio nach außen wie ein kommerzielles Produkt ohne Würdigung einzelner Mitarbeiter.

Quellenverzeichnis

Literaturverzeichnis

Colazo, J. / Fang, Y. (2009)	The impact of license choice on open source software development activities. Journal of the American Society for Information Science and Technology, New York: o.V.
Egan, S. (2005)	Open Source Messaging Application Development: Building and Extending Gaim (Expert's Voice in Open Source), New York: Springer Verlag
Fang, Y./ Neufeld, D (2009a)	Understanding Sustained Participation in Open Source Software Projects, o.O: Journal of Management Information Systems
Frendo T. (2009)	Factor of Success in Open Source Software, Master Thesis
H. Klapf/ R. Pölsch (2010)	Studien Open- Commons- Region Linz, Fakten Perspektiven, Maßnahmen, „Kriterienkatalog zur Identifikation von Open Source Einsatzgebieten“
S. Schaffert/ D. Wieden-Bischof (2009)	Erfolgreicher Aufbau von Online-Communitys: Konzepte, Szenarien und Handlungsempfehlungen, Salzburg: NewMediaLab

Schweik, C. M. et al (2009)	Factors Leading to Success or Abandonment of Open Source Commons: An Empirical Analysis of Sourceforge.net Projects, Amherst: National Center for Digital Government

Internetquellen

Buytaert, D. (o.J.)	Drupal, http://drupal.org/node/228222, Abruf: 04.01.2012
Buytaert, D. (o.J.)	Drupal (Community & Support), http://drupal.org/getting-involved, Abruf: 05.01.2012
ChipOnline (o.J.)	http://www.chip.de/bildergalerie/Top-50-Open-Source-Downloads-September-2011-Galerie_47813697.html, Abruf: 26.12.2011
Debian GNU (2011)	Debian GNU/Hurd, http://www.debian.org/ports/hurd/, Abruf: 04.01.2012
Debian GNU (2011)	GNU/ Hurd/ User Pages, http://www.gnu.org/software/hurd/user.html, Abruf: 17.12.2011
DistroWatch.com (2004)	DistroWatch Page Hit Ranking, http://distrowatch.com/dwres.php?resource=popularity, Abruf: 07.01.2012
Foswiki (o.J.)	Foswiki, Project Culture, http://foswiki.org/Community/ProjectCulture, Abruf: 03.01.2012
FosWiki (o.J.)	http://foswiki.org/About/WhyThisFork?redirectedfrom=Home.WhyThisFork, Abruf: 04.01.2012

Idealware (2010)	Comparing Open Source Content Management Systems: WordPress, Joomla, Drupal and Plone, http://www.idealware.org/reports/2010-os-cms?key=45152768, Abruf: 06.01.2012
Jahia Solutions Group SA (2002)	Jahia, http://www.jahia.com/cms/home/about-us/business-model.html, Abruf: 04.01.2012
Jive Software (o.J.)	OpenFire Community, http://www.igniterealtime.org/projects/openfire/, Abruf: 03.01.2012
Jive Software (o.J.)	OpenFire, http://community.igniterealtime.org/community/developers/openfire_dev, Abruf: 29.12.2011
Joomla (o.J.)	http://www.joomla.org/about-the-joomla-project/sponsorships/development-sponsors.html, Abruf: 26.12.2011
MTV Networks (2011)	MTV, http://www.mtv.co.uk, Abruf: 07.01.2012
o.V. (2003)	XMPP (Extensible Messaging and Presence Protocol), http://searchdomino.techtarget.com/definition/XMPP, Abruf: 08.01.2012
o.V. (2006)	Method for Qualication and Selection Of Open Source software (QSOS), http://www.qsos.org/download/qsos-1.6-en.pdf, Abruf: 03.01.2012
o.V. (o.J.)	Pidgin – Alternative oder Ergänzung, http://www.pc-magazin.de/ratgeber/multi-protokoll-client-1048848.html, Abruf: 03.01.2012
o.V. (o.J.)	http://t3n.de/opensource/projects/, Abruf: 14.01.2011
o.V. (o.J.)	http://web.archive.org/web/20061201073806/www.sueddeutsche.de/kultur/artikel/826/54772/, Abruf: 30.12.2011

o.V. (o.J.)	Developer Summit, http://uds.ubuntu.com/, Abruf: 09.01.2012
Oracle Corporation (2010)	The Meanings and Motivations of Open-Source Communities, http://java.sun.com/developer/technicalArticles/javase/opensource_phipps, Abruf 16.12.2011
Radcliffe, M. (o.Ja)	The Open Source Definition, http://opensource.org/docs/osd, Abruf 27.12.2011
Radcliffe, M. (o.Jb)	http://opensource.org/licenses/alphabetical, Abruf: 29.12.2011
Shreves, R. (2010)	2010 Open Source CMS Market Share Report, http://www.waterandstone.com/book/2010-open-source-cms-market-share-report, Abruf: 06.01.2012
Source forge (o.J.)	http://sourceforge.net/, Abruf: 16.12.2011
Source Forge (o.J.)	Xoops, http://sourceforge.net/blog/podcast-xoops, Abruf: 05.01.2012
The Ubuntu Foundation (2011a)	Canonical and Ubuntu, http://www.ubuntu.com/project/canonical-and-ubuntu, Abruf: 29.12.2011
The Ubuntu Foundation (2011b)	New Ubuntu Foundation Announced, http://www.ubuntu.com/news/UbuntuFoundation, Abruf: 08.01.2012
Thoma, J. (2011)	Entwickler programmieren, um zu lernen, http://www.golem.de/1102/81153.html, Abruf: 07.01.2012
XOOPS Awards and Recognitions (2010)	Xoops, http://www.xoops.org/modules/wfchannel/index.php?cid=28, Abruf:03.01.2012
yeebase media GmbH (2005)	Open Source Software Top 100, http://t3n.de/opensource/top100/, Abruf 03.01.2012

Development of a Model Evaluating the Maturity of Open Source Software

Thesis

submitted on 5th July 2012

School of: Business

Program: International Business Information Management

Course: WWI 2009 I

by

Franziska Gorhan

Juliana Hettinger

Juliane Schulz

Maren Wolter

BW Cooperative State University Stuttgart:

Prof. Dr. Kessel

Table of Contents

List of Abbreviations IV

List of Figures V

List of Tables V

1 Introduction 1

1.1 Problem Definition 1

1.2 Objectives 2

1.3 Approach and Methodology 2

2 Definition of Important Terms 4

2.1 Open Source Projects 4

2.2 Maturity Models in general 6

2.3 CMMI as a Standard 7

2.3.1 Idea of CMMI 7

2.3.2 Functionality of CMMI 8

3 Existing Maturity Models 9

3.1 Open Source Maturity Model from Navica 9

3.2 Open Source Maturity Model from Capgemini 11

3.3 Open Business Readiness Rating 12

3.4 Qualification and Selection of Open Source Software Model 15

3.5 Summary of Basic Maturity Models 18

4 Developed Maturity Model 21

4.1 Selected Criteria 21

4.2 Weighting Factors for the Calculation of the Overall Maturity Score 26

4.3 Introduction of the New Maturity Model 28

5 Maturity of Appropriate Case Studies 30

5.1 Selection of Case Studies 30

5.2 Result of Case Studies 31

5.3 Maturity of the Example 32

6 Critical Reflection on the Maturity Model Introduced 36

7 Conclusion 38

7.1 Summary 38

7.2 Outlook 39

Appendix 40

Lists of References 61

List of Abbreviations

CMM	= Capability Maturity Model
CMMI	= Capability Maturity Model Integration
FLOSS	= Free/Libre Open Source Software
FSF	= Free Software Foundation
GNU	= General Public License
OSD	= Open Source Definition
OSS	= Open Source Software
OSMM	= Open Source Maturity Model
QSOS	= Qualification and Selection of Open Source Software Model
SEI	= Software Engineering Institute

List of Figures

Fig. 1: Steps of QSOS 15
Fig. 2: Ready For Business Model 28
Fig. 3: Rating of Case Studies 31

List of Tables

Table 1: Maturity Levels CMMI 8
Table 2: Default Score List 10
Table 3: OpenBRR Categories to be Assessed and Evaluated 13
Table 4: Weighting for Criteria 16
Table 5: Example for Evaluation 17
Table 6: Summary Maturity Models 19
Table 7: Maturity Levels 29

1 Introduction

1.1 Problem Definition

When content is shared with and created by several people in the world the term Open Source (OS) is often mentioned. The purpose of Open Source is to gather knowledge and experience of different people. Large corporations like IBM are investing in Open Source Software (OSS). Before the expression *OS* became a broadly adopted concept, developers and producers used a diversity of terms to describe this concept. With the evolving internet though, the term Open Source Software became accepted.[1]

Open Source is an approach where software developers promote free redistribution and access to an end product's design and besides that implementation details (binary code and source code). This code can be further distributed or individually modified. Making the source code available to the general public allows a self-enhancing variety of production models, communication paths, and interactive communities. The modified source code is subject of numerous appropriate public discussion forums and is reviewed by many other developers. This leads to a general high quality of the so-developed software. When accessing the code there might be a little or no cost recovery fee. An example of very successful OS-developments in general is the Linux operating system and, more recently, the open operating system Android for mobile devices (smartphones, tablets, etc.). Furthermore, Gnome, Apache, Firefox, Thunderbird, and OpenOffice are well known software that is developed by communities. This kind of software is increasingly being applied in other fields of endeavor, such as biotechnology.[2]

Since OS-projects almost always comply with international standards, there are many IT service providers with the appropriate skills, offering their services around the solutions.

Software and community service quality have noteworthy effects on user gratification. Furthermore, usefulness, ease of use, and reliability are some of the major factors that OSS experts should pay attention to in order to improve OSS quality. Especially nowadays: OSS is no longer used solely by computer developers; the number of nontechnical and new computer users is growing at a rapid pace, highlighting the need of understanding and addressing their requirements and expectations.[3]

There is a general problem concerning OSS. Besides its easy handling it is hard to measure the maturity of OSS. There are theoretical models which have complicated measuring

[1] Cf. van den Berg, K. (2005), p. ii
[2] Cf. Van Laak, K. (2012)
[3] Cf. Raza, C. (2012)

methodologies in order to find out the ease of treatment and usefulness. Companies and specialists do not trust theoretical patterns but practical oriented models that are simpler in their application. This challenge needs to be addressed.

In order to understand OSS better, maturity models are designed which can be used as a tool to find the right software package to meet the single user´s desires. With these models an evaluation about open software is processed. The evaluation is handled by using different criteria.[4]

1.2 Objectives

The willingness of a company to use OSS is a good starting point. But there are still a few steps to make until effective usage. First, appropriate software solutions have to be identified and checked for their suitability, for example concerning the required functionality. The aim of this project paper is to design a model that determines the maturity level as well as the suitability of an OS-product. This maturity model will be developed in form of an Excel tool.

The demand for easy usable products is growing, and an appropriate, end-user friendly measurement model is appreciated nowadays. The tool will also be applied to chosen case studies. The user-friendly Excel implement will be prepared in order to quickly enable the determination of the maturity of an OSS.

1.3 Approach and Methodology

The structure of this project paper is divided into seven chapters. The composition of the project paper is equal to the development of the empirical research project.

In the first part, the topic will be introduced by explaining the challenges of open source software and highlighting its importance. Also, the aim of this paper is defined.

In the next chapter there is an outline of the basic knowledge in form of elementary definitions of the most important terms. Besides this, a common maturity model, called Capability Maturity Model Integration (CMMI), which can be used by organizations for enterprise-wide process improvement is introduced which is in many respects a standard and basis.

---

[4] Cf. van den Berg, K. (2005), p. ii

Chapter 3 takes a closer look at what kind of maturity models are provided by the market. The following four maturity models will be discussed in more detail: Open Source Maturity Models from Navica and from Capgemini, the Open Business Readiness Rating, and the Qualification and Selection of an Open Source Software Model.

In chapter 4, a new model will be developed and presented. The methodology of this model is described precisely in combination with 16 criteria, which are required in order to determine a maturity level of OSS.

On the basis of the developed model, a case study will be processed. Four open source projects will be observed and examined and finally their maturity level will be determined by using the new maturity model. Those models will be the OSS Android, Mozilla Firefox, WebOS, and the VLC media player.

In the last part of the paper, there will be a critical reflection of the project, which mainly refers to the process of the empirical research. Afterwards, a conclusion will be provided. Closing up, there will be an outlook of how the recommendation, which was created during the analysis of the case studies, will influence the future market.

2 Definition of Important Terms

2.1 Open Source Projects

The term Open Source refers to a software development method.[5] Moreover, software that is distributed under terms complying with the Open Source Definition (OSD) is described as Open Source Software. OSD is not a license itself. It is a document maintained by the Open Source Initiative including criteria concerning a software product's way of distribution and licensing. Software products that comply with these criteria are said to be open source and OS projects are the projects in which this kind of software is developed, whereby the license prescribes the terms of use.[6]

Coming to their lifecycle, OS projects begin with individuals or groups that are working on a tool based on an initial idea. After the first release of the developed OSS the amount of contributors and thereby the software's community grows.[7] Then the OSS is able to benefit from the code improvements provided by all dispersed contributors. Unlike proprietary software engineering, OSS development has no fixed end date or number of participants.[8] In most cases, the leadership and governance of OS projects remain in one central team that coordinates the development and decides on the features to be included in further distributions of the software.[9] From a legal perspective, the free provision of OSS is a gift, whereby the software is provided without liability and warranty, even in case of claim and danger.[10]

According to the Open Source Definition, the following criteria must be fulfilled by the distribution terms of OSS.[11]
The free redistribution or resale must be possible without having to pay fees or royalties to original authors of the software.[12] According to this criterion, OSS can be sold and is not necessarily free software.[13] Furthermore, a developed program must include source code and allow the distribution in a compiled form as well as in source code. Derived works and

[5] Cf. Open Source Initiative (w.y.)
[6] Cf. Feller, J./Fitzgerald, B. (2002), p. 12 et seq.
[7] Cf. Rossi, M.A. (2006), p. 18 et seq.
[8] Cf. Raymond, E.S. (2001), p. 21 et sqq.; cf. ibidem, p. 40 et seq.
[9] Cf. Kavanagh, P. (2004), p. 203 et sqq.
[10] Cf. Paul, A.G. (2006), p. 69 et seq.; cf. de Vuyst, B./Fairchild, A. (2007), p. 334 et seq.
[11] Cf. Open Source Initiative (w.y.)
[12] Cf. ibidem
[13] Cf. Feller, J./Fitzgerald, B. (2002), p. 13

modifications must be allowed in the license of the OSS and their distribution must be possible under the same terms as with the original software. Concerning the integrity of the author's source code, the distribution of software that is based on modified source code must be explicitly permitted in the software's license. Furthermore, the license must not include discrimination against persons or groups and discrimination against fields of endeavor, meaning that nobody is restricted from using the program in specific fields of endeavor. Concerning the distribution of licenses, to all to whom the software is redistributed, the rights that were attached to it apply. Additionally, the license must not be specific to a product, as it does not depend on software being part of a specific distribution. Other software distributed together with the OSS must not be restricted thereby and the OSS license must be technology-neutral.[14]

To ensure the compliance of OSS with the norms and expectations of the Open Source Initiative, all licenses have to pass a public review process for approval.[15] The more than 60 approved licenses can be separated in the categories: copyfree, strong copyleft or weak copyleft. To ensure that OSS remains open source, the strong copyleft attribute determines the release of derivative works under the same license as of its ancestor.[16] In contrast, the copyfree attribute does not include restrictions about the licenses for derivative works and can be commercialized through relicensing of derivative works. The weak copyleft license is a compromise between these two types of licenses. It allows software components to be part of or linked with proprietary software without becoming proprietary itself and is often used for software libraries.[17]

The Berkeley Software Distribution is the most widely used permissive OSS license including the copyfree attribute. This license allows for copying, modifying and redistributing OSS, whereby the copyright note of the source program must be included.[18] The advantage of the Berkeley Software Distribution license is that it enables the combination of open source and commercial software without making the whole combined product open source.[19]

The predominantly used hereditary OSS license including the strong copyleft attribute is the General Public License. This license was designed by the Free Software Foundation (FSF) to ensure that derived software versions are also licensed with the General Public License

[14] Cf. Open Source Initiative (w.y.)
[15] Cf. ibidem
[16] Cf. Meeker, H.J. (2008), p. 22 et seq.
[17] Cf. Wheeler, D.A. (2007)
[18] Cf. Meeker, H.J. (2008), p. 43 et seq.
[19] Cf. Rosen, L. (2005), p. 179

and that all rights are transferred to the FSF to prevent new versions from being licensed differently.[20]

The Lesser General Public License (LGPL) is the most widely used weak copyleft license. According the LGPL, software components that are licensed with the LGPL can be linked with programs, which are not released under this license, no matter if they are proprietary of free software.[21]

In the evolution of Open Source, divergences aroused concerning the terms “Open Source” and “free software”. In contrast to free software that stands for being free of charge and providing liberties to users, OS allows the access to the source code, whereby the individual licenses prescribe the terms of use. In order to avoid the discussion of the differences, the term Free/Libre Open Source Software (FLOSS) was introduced and will be used in the following chapters of this paper.[22]

2.2 Maturity Models in general

The demand and need for companies finding a way to assess different FLOSS solutions referring to their functionality as well as expected quality standards is steadily growing as a result of the vast and growing range of available FLOSS solutions. Besides evaluating a FLOSS product’s maturity, an Open Source Maturity Model’s (OSMM) intention also considers a comparison of the software with commercial alternatives as well as its correspondence to specific business but especially IT requirements.

Therefore, the Maturity Model provides a guideline how a FLOSS product should be assessed which besides FLOSS product analyses also includes the review of the referring company and its current IT issues.[23]

---

[20] Cf. Meeker, H.J. (2008), p 29 et seq.
[21] Cf. Free Software Foundation (2007)
[22] Cf. Stallman, R.M. (2002), p. 58
[23] Cf. Russo, B. et al. (2010), p. 303 et sqq.

2.3 CMMI as a Standard

2.3.1 Idea of CMMI

For the development of software a process has to be conducted. Furthermore, in order to fulfill this process efficiently, effectively and also to reach business success, a standard has evolved. The standard is called Capability Maturity Model Integration (CMMI), which is a framework for organizations to improve the ability for building and maintaining a quality product as well as services.[24] The idea behind CMMI was to create one single model which can be used by organizations for enterprise-wide process improvement.[25] The Software Engineering Institute (SEI) at the Carnegie Mellon University believed that there is not one single model which is the very best suited one for every organization and industry. This is because the various models have different focus.[26] Therefore, the SEI published the CMMI in 2002, in order to offer an integrated and comprehensive model in the area of process improvement.[27]

The CMMI is a further developed version of the Capability Maturity Model (CMM) for software to an integrated offer. The move away from a process improvement strictly for software has been seen critical. It is argued though that the concepts of the CMM for software were simply upgraded. The decennial experience with CMM was used to create the new model CMMI.[28] The new model enables guidance for the software development processes as well as integrated systems. The coordination of multidiscipline activities in order to build a project is now possible without using multiple models, but simply the CMMI.[29]

Nowadays, when an organization is interested in process improvement, it has to look at the available models and find out which fits best to the business needs. The CMMI is chosen as a basis for this paper, because it provides the basic understanding of process improvement and is considered as a broad standard.[30]

---

[24] Cf. Belcher, J.M. (2012)
[25] Cf. Kasse, T. (2004), p. 28 et sqq.
[26] Cf. Kenett, R.S./Baker, E. (2010), p. 65 et sqq.
[27] Cf. Myerson, J.M. (2007)
[28] Cf. Kasse, T. (2004), p. 140 et sqq.
[29] Cf. ibidem, p. 32 et sqq.
[30] Cf. Kenett, R.S./Baker, E. (2010), p. 78 et sqq.

2.3.2 Functionality of CMMI

The CMMI is structured into five maturity stages, which are in an ascending order: initial, managed, defined, quantitatively managed, and optimizing. An overview with the according description of each level is given in table 1.

Maturity Level	Name	Description
1	Initial	Processes are done ad hoc and are also chaotic.
2	Managed	A basic infrastructure for project management exists. The performance of the projects is consistent. Processes are reactive.
3	Defined	There is a standard process for performing projects and for the management of processes. The project management is proactive.
4	Quantitatively Managed	Quantitative management of the processes is implemented. The quantitative process performance and quality goals as well as the objectives are established.
5	Optimizing	The focus is laid on improving the processes of the organization continuously. The capability of the processes of meeting quality and process performance goals and objectives is evaluated.

Table 1: Maturity Levels CMMI[31]

The model is constructed as an integrated model. If one would like to reach a certain level of maturity one has to fulfill all requirements of that certain level and also all requirements of the levels below. This means the levels below are integrated in the certain level. In order to meet the requirements of the level, the goals of each project area have to be fulfilled. A project area is a group of various practices and activities which are performed collectively in order to reach an objective. For example in level 2 project planning is one of the project areas which needs to be fulfilled. In general, the process areas can be grouped in four different categories: process management, project management, engineering and support.[32] An overview of all maturity levels with the according project areas can be found in appendix 1, page 41. Level 5 means an organizations reaches full maturity, but this is achieved only rarely by approximately 2% of the organizations.[33]
The structure of CMMI will be the foundation for the new maturity model, meaning that one firstly looks at the given criteria and subsequently evaluate them. Furthermore, the result delivered by the new model will be structured in maturity levels as well.

---

[31] Cf. Kenett, R.S./Baker, E. (2010), p. 70 et sqq.; cf. Kulpa, M.K./Johnson, K.A. (2003), p. 18 et sqq.
[32] Cf. w.a. (2012)
[33] Cf. Persse, J.R. (2001), p. 3

3 Existing Maturity Models

For the assessment of software, different Capability Maturity Models can be used. As the characteristics of FLOSS products differ from commercial software, appropriate maturity models have to be looked at. Officially, the first FLOSS quality and maturity models emerged in the years between 2003 and 2005. Among the so called “first generation FLOSS quality models”[34] the most common ones shall provide the basis for the development of a new well thought-through maturity model introduced in Chapter 4. Therefore, the Open Source Maturity Models from Navica and from Capgemini, a model called Open Business Readiness Rating, and the Qualification and Selection of Open Source Software Model will be introduced briefly, describing their effectiveness and evaluating their functionality.[35]

3.1 Open Source Maturity Model from Navica

The Open Source Maturity Model from Navica is based on a methodology using scores for assessing FLOSS. Maturity scores allow the comparison of different OS-products. Nearly 85% of OS users demand tools, descriptions, assessments, robust support and documentation besides just the original product in order to understand and be able to use the software as efficient as possible. Proprietary software usually contains this additional information while it is not available when development is still in progress.[36]
Due to this fact, the Navica maturity model concentrates also on those additional components and elements besides the software’s code. Within the three phases of Navica’s model, which will be explained later on, there are six criteria on which maturity is calculated and which have to be initially introduced:

- product software
- support
- documentation
- training
- product integrations
- professional service[37]

In the first of the three phases, the user has to assign a maturity score to each criterion. The score range is from 0 to 10 points and it represents how well the element meets user

[34] Cf. Norsk Regnesentral (2011), p. 75
[35] Cf. Russo, B. et al. (2010), p. 303
[36] Cf. ibidem, p. 305
[37] Cf. ibidem

requirements.[38] Within the second phase, the user is prompted to assign weighting factors (importance of a product). There is a default score list fitting to the criteria shown in the following table:

Criterion	Software	Support	Documentation	Training	Integration	Professional services
Weighting factor	4	2	1	1	1	1

Table 2: Default Score List

Table 2 shows the correlation between the weighting factors and the criteria. The first row includes the six criteria while the second row contains the matching weighting factors. These weighting factors can be changed individually but their sum always has to be 10.

Third phase: After assigning the points to every criterion and looking at the default weighting factors a calculation has to be made by multiplying every maturity score with the fitting weighting factor. The sum of this calculation gives an overall product maturity score that ranges between 1 and 100.[39]

Evaluation

There are onetime scores for each criterion and also one global score for the whole FLOSS product.[40] This global score reveals maybe in general that the software is good but one criterion score might be still improvable. Therefore, this model provides an indication for an improvement of one or more singular parts of the assessed software.

Nevertheless, there are difficulties with this model. The ratings for both, the maturity score and the final calculation are not accurately defined. The maturity score is assigned individually and subjectively by every user. Therefore, the score is not objective but rather influenced by the opinion and personal experiences of a person (tester). The scale between 0 and 10 should have a precise description of what each number represents.

Furthermore, the final calculation shows a number between 1 and 100 but at the end there is no scale of declaration what each number means or predicates. When using a maturity model a recommendation about the assessed software should be available in order to be able to evaluate them. This is something that definitely needs to be implemented differently in the newly developed model.

---

[38] Cf. Russo, B. et al. (2010), p. 305
[39] Cf. ibidem, p. 306
[40] Cf. ibidem

This Navica maturity model is easy to custom and covers all significant steps needed to define the maturity of considered FLOSS-products. Hence, this model can be easily used as a foundation, while it can also be used in a relatively short time frame and is easily accessible for the user due to the fact that there are only six criteria needed. But just that is not enough for the new model which is to be developed. Therefore the following model from Capgemini will also be looked at.

3.2 Open Source Maturity Model from Capgemini

Within commercial environments, FLOSS products can be evaluated and implemented according to the systematic Open Source Maturity Model from Capgemini, which is a global IT service and management consulting company. In order to meet all the IT requirements of a company that plans to make use of a FLOSS product, the maturity of the product must be determined, the product’s match to business requirements must be assessed and a comparison with commercial alternatives must be made.[41]

The OSMM procedure developed by Capgemini includes seven steps as explained in the following. In the first step, product research takes place and appropriate solutions are roughly selected. Thereafter, product indicators are used for product scoring. For this purpose, the four product indicator groups: *product*, *integration*, *use* and *acceptance* are used. The indicators of the product group focus on the functions of the product, whereas the integration indicators are used to measure the integration possibilities of the analyzed FLOSS product with other components. The everyday support of a FLOSS product is measured with the use of indicators and the perception of the product inside the Open Source community is assessed with the help of indicators contained in the acceptance group. In the third step, application indicators are used to assess the FLOSS products with regards to user’s future and present needs. The applied measures comprise *performance*, *usability*, *security*, *interfacing*, *support*, *reliability*, *reporting*, *proven technology*, *platform* and *vendor independence*, *training*, *implementation*, *staffing*, *administration*, and *advice*. The subsequent fourth step is used to interview the customer to identify the importance of individual application indicators. After that, all application indicators are scored with the help of the

[41] Cf. Russo, B. et al. (2010), p. 303 et sqq.; cf. Capgemini (2012)

customer and in the sixth step score cards are determined for the FLOSS products and the final selection is made. The selected product is finally evaluated in the seventh step.[42]
For the individual indicators used in the scores in step two, three and five, values between one and five can be given. Thereby, scores below three indicate a negative assessment result and scores above three a positive one. If one of the indicators is not applicable for assessing a FLOSS product, the score of this indicator is set to three, whereby it does not affect the scoring. Thus, FLOSS products with an overall score below three are excluded from the selection.[43]
For some of the described indicators the data collection is done in three different time periods. The time between the first and the second data collection comprises six months and the third one is done two years after the second collection. As soon as the data collection is completed, the scores can be combined to one final score indicating the sustainability of a FLOSS product.[44]

Evaluation

On the one hand the Open Source Maturity Model from Capgemini is a structured approach including predefined steps and a variety of indicators. On the other hand, this model does not enable short-term decisions due to the fact that the data collection stretches over two and a half years. Furthermore, there is a lack of information about this model. In step four, for example, the customer is interviewed about the importance of individual application indicators. However, no information about the consequences of this interview regarding the weighting of individual criteria could be found.
The long data collection period seems to prevent quick decision making but ensures a thorough basis for decisions. Thus, Capgemini's model calls attention to the meaningfulness of data. In addition to that, this model includes numerous criteria, implying its thoroughness as well.

3.3 Open Business Readiness Rating

The Open business readiness rating (OPENBRR) was created in 2005 by the SpikeSource Centre and Intel Corporation, which have specified four crucial requirements that are aimed to be addressed by this model. Firstly, the model must reveal all positive as well as negative

[42] Cf. Russo, B. et al. (2010), p. 304 et seq.; cf. NRC FOSS (2009)
[43] Cf. Russo, B. et al. (2010), p. 304 et seq.; cf. NRC FOSS (2009)
[44] Cf. Russo, B. et al. (2010), p. 304

characteristics the software has in order to provide a clear overview. Secondly, it must have an easy comprehensible assessment process and score system with an easy to understand terminology used throughout the model. Thirdly, the model must be flexible in order to guarantee adaptability to rapid changes in the IT sector. Last but not least, it must be consistent across all users.

Besides those requirements the 4 phase assessment process included in this model is of high importance. In the first phase of **Initial Filtering** a rough estimation of quantitative and qualitative characteristics of the software is done such as implementation language or licensing/legal situation of the software. The phase of **Target Usage Assessment** includes both category as well as metrics weighting. All 12 categories to be assessed are listed and shortly described in table 3.

Assessment Category	Description
Functionality	How well will the software meet the average user's requirements?
Usability	How good is the UI? How easy to use is the software for end-users? How easy is the software to install, configure, deploy, and maintain?
Quality	Of what quality are the design, the code, and the tests? How complete and error-free are they?
Security	How well does the software handle security issues? How secure is it?
Performance	How well does the software perform?
Scalability	How well does the software scale to a large environment?
Architecture	How well is the software architected? How modular, portable, flexible, extensible, open, and easy to integrate is it?
Support	How well is the software component supported?
Documentation	Of what quality is any documentation for the software?
Adoption	How well is the component adopted by community, market, and industry?
Community	How active and lively is the community for the software?
Professionalism	What is the level of the professionalism of the development process and of the project organization as a whole?

Table 3: OpenBRR Categories to be Assessed and Evaluated[45]

As for category weighting, the user must list seven out of twelve categories with number 1 being the most and number 12 the least important. As for metrics weighting, the user has to rank each metric within the referring category, whereas each metric's percentage of perceived importance adds up to a 100% for that category. Phase three, **Data Collection**

[45] Contained in: OpenBRR.org (2005), p. 7 et seq.

and Processing collects data for each category's metrics and calculates the applied weighting for each. As the last phase, **Data Translation** is the one in which the BRR score is calculated based on category ratings and weighting factors.[46] For a practical implementation, the BRR model provides a set of pre-defined metrics for each category, which are not to be seen as the definite set, but an initial one that allows for continual improvement and can be found in the OpenBRR whitepaper. [47]

Evaluation

As a standardized assessment model, OpenBRR allows consistency when comparing different OSS from different domains. A transparent assessment enables an improved systematic implementation accelerating the OSS's adoption. At the same time it is also open and flexible. This allows its extension without serious problems especially due to influencing changes within the IT sector.[48] When the model was introduced in 2005, representatives made clear that membership to the OpenBRR corporate community is not open to all but on an invitation-only basis to ensure trusted participants only. This, as well as the model's openness ensures a special trustworthiness in its components. The model was also introduced as highly private so that members can discuss sensitive issues without getting them out to the general public. As well, members can share information anonymously and they intended to introduce a monthly newsletter, Wiki Threads, private discussion forums, e-mail lists.[49] The evaluation process takes a lot of time though because one has to consider what an average user's standard feature set might be which then must be applied strictly. Additionally, each piece of the test-software will have to be installed and testing should also be done by a real end user afterwards.[50] Having a look at the official webpage, it says that there is not yet a thriving community. Neither is the website of OBRR providing the whitepaper or templates for the evaluations, it only has a main page, no menu or links, and no e-mail or contact details from authors.[51] This is a sign that the model as well as its development has been frozen for a while. An evaluation template from 2005 was found through research in articles and having a deeper look into it, it seems hard to understand.[52] Still it remains unclear if this is an official template e.g. developed by the OpenBRR community or if it was developed by someone from the outside. The general web does not find many cases of evaluated projects using the method. Therefore, it can be assumed that the model has not seen a wide adoption and cannot be considered a success as it is

---

[46] Cf. Russo, B. et al. (2010), p. 308 et seq.
[47] Cf. OpenBRR.org (2005), p. 17
[48] Cf. Russo, B. et al. (2010), p. 308 et seq.
[49] Cf. Ziff Davis Enterprise Holdings Inc. (2012)
[50] Cf. Berry, M (2009)
[51] Cf. Network Solutions, LLC (2005)
[52] Cf. w.a. (2005a)

basically dead. Otherwise, one success story can be found: the Open University's use of OpenBRR for choosing a Virutal Learning Environment in 2006.[53]

The OpenBRR whitepaper can only be found in "non-official" places meaning no pages being from the official creators of the model.[54] All in all, the idea behind the model seems reasonable as well as practicable but it seems not to have established as well which may have a number of reasons including the few ones mentioned above. Furthermore, it seems not to provide enough specificity and effectiveness to make a firm decision and metrics may have to be extended.

3.4 Qualification and Selection of Open Source Software Model

Another model which has established is the Qualification and Selection of Open Source Software Model (QSOS). The objective of this model is to qualify, select as well as compare FLOSS-products. The model delivers information in order to differentiate FLOSS products in regard to technology and functionality. Furthermore, it provides a framework for an efficient risk management process.[55] QSOS consists of four iterative steps, which have to be applied consecutively. The steps are shown in figure 1.

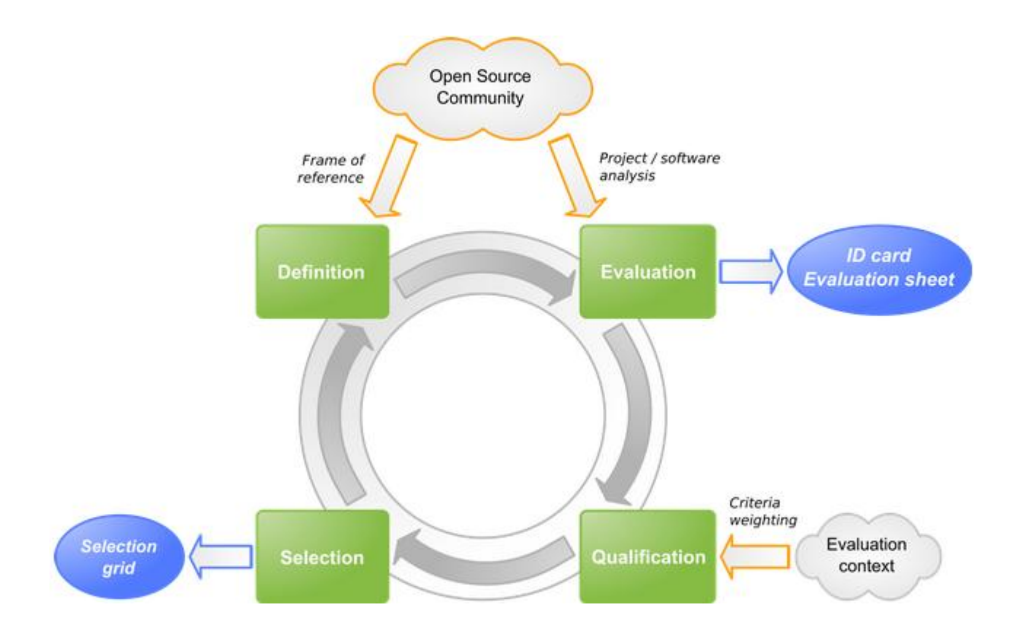


Fig. 1: Steps of QSOS[56]

[53] Cf. Haaland, K. et al (2010), p. 4
[54] Cf. Arjona, L. (2012)
[55] Cf. Russo, B. et al (2010), p. 307
[56] Contained in: Origin, A. (2006), p.8

In the first step, called **definition**, the frames of references in regard to licenses, communities and software families are clarified. For example, concerning the communities it can be differentiated amongst other criteria between a group of developers and an organization of developers. In the second step, the software is **evaluated** according to the given criteria. In general, there are three topics regarded: the functionality, the risk for the user and the risk for the service provider. The specific criteria are explained below. For the evaluation of the model three categories can be distinguished:

not covered = 0,

partially covered = 1 and

completely covered = 2.

In step three, the evaluation can be tailored to the specific user's needs by **weighting** the criteria. There are three levels of relevance of each criterion. The level is then converted into a numeric value as shown in the table 4 below. The positive and negative sign represent the impact for the user.

Relevance	Weight
Irrelevant criterion	0
Relevant criterion	+1 / -1
Critical criterion	+3 / -3

Table 4: Weighting for Criteria[57]

Finally in step 4, the different available software products can be compared and the best one gets **selected**. The result is based on the steps before and in this step a ranking of the possible FLOSS products can be done. There are two types for evaluating the products: strict and loose selection. When the strict selection is chosen, the score of a relevant criterion has to be at least 1 and for a critical criterion it has to be 2, otherwise the product is eliminated from the result list of possible FLOSS products. The loose selection uses the table for weighting shown above in step 3. Hereby, the products are not eliminated immediately, but they are classified. The recommendation by QSOS is to consider the top 2 – 5 products, which meet the qualification criteria.[58]

[57] With modifications taken from: Origin, A. (2006), p. 32
[58] Cf. w.a. (2006)

Criteria:

From the three high level criteria, which are the functionality, the risk for the user and the risk for the service provider, the focus lies on the risk of the user for this paper. Therefore, only the according criteria for the user's risk are regarded in detail, while the criteria for the service provider and functional coverage are not regarded any further. The criteria are set up in a tree hierarchy.[59]

User risk: This category regards the risk which evolves when the user adopts the software. On the first level it is differentiated between five categories, which are split up into sub categories as indicated with the brackets behind each top level category.

1. Intrinsic durability [Maturity, Adoption, Development Leadership, Activity, Independence of Developments]
2. Industrialized solution [services, documentation, quality assurance, packaging]
3. Integration [packaging]
4. Technical adaptability [modularity, byproducts]
5. Strategy [license, copyright owners, modification of source code, roadmap, sponsor, strategical independence]

To define the intrinsic durability of a FLOSS product the first category regarded is the maturity. Within the maturity there are more sub categories: age, stability, history known problems and forked probability. All these categories have to be evaluated. For example, the maturity of the age is determined in the following way. The age of the FLOSS product simply needs to be defined and then put in the fitting category of table 5 shown below.

Age	0	1	2
	Less than 3 months	Between 3 months and 3 years	More than 3 years

Table 5: Example for Evaluation[60]

There is a list available for all criteria and sub criteria with the according evaluation. The list for the user risk can be found in appendix 2, page 42.

Evaluation

The model QSOS has already been reviewed in the literature and the following results can be stated. To begin with, there are many advantages. First of all, the model provides the result in form of an absolute score. This is supported by the small score range from 0 to 2.

[59] Cf. Origin, A. (2006), p. 15 et sqq.
[60] Contained in: ibidem, p. 16

On the one hand, this leads to less variance in results. On the other hand, the evaluator may not find the right score in such a small range and therefore information may get lost.[61] Secondly, there is a clear list of criteria available. Finally, the model can be tailored to the user's needs by adjusting the importance of the criteria.

On the contrary, the following disadvantages have to be taken into account. Firstly, the model is very complex as it is set up in a tree hierarchy and many criteria are available. For the application of the model this means it is time-consuming. Furthermore, the data for the evaluation of the criteria has to be available. If it is hard to find the information, one has the option to put a question in a forum. This would then require the help of the community to find the information. For this model five criteria have been identified, where it is unlikely to find data about criteria like: history of known problems, fork probability, management style, developer identification and independence of developments. Moreover, the criteria may be ambiguous for the user.

Considering all points, the model has a good structure with an absolute score as a result, but the choice of criteria is too complex for an easy use of the model.[62]

3.5 Summary of Basic Maturity Models

All of the models introduced are based on manual work including templates and evaluation forms making the data gathering for evaluation and the actual evaluation a manual work process. The maturity models from Navica, Capgemini, the OpenBRR and the QSOS model are building the basis for the new model. In this section the strengths and weaknesses of the regarded models will be measured and the best approach for the new model will be developed. Table 6 shows an overview of the strengths and weaknesses of each model.

---

[61] Cf. Deprez,J./Alexandre, S. (w.y.), p. 8 et seq.
[62] Cf. Ortega, F./Izquierdo, D./ Coca, P. (2010), p.18 et sqq.

	Navica	Capgemini	OpenBRR	QSOS
Strengths	three predefined steps	structured approach including pre-defined steps	number of criteria evaluated stays realistic	good structure
	manageable number of criteria	variety of criteria	easy to extend	popularity
	short time frame and easy accessible for the user (6 criteria)		flexible	only 3 weighting factors
	result in form of an absolute score			result in form of an absolute score
Weaknesses	weighting for result is difficult to approach	no short-term decisions → data collection stretches over 2.5 years	Evaluation takes a lot of time	complex and time consuming to apply
	low number of criteria can mean a risk for not covering all aspects	lack of information about this model	no support for the model, lack of information	many criteria which have to be evaluated
		too many criteria for easy usage	not widely adopted	

Table 6: Summary Maturity Models

The Open Source Maturity Model from Navica is seen as a good approach, because it is easy to use and covers all important steps needed to define the maturity. Therefore, the model to be developed can use this as a foundation. In the model from Navica there are three steps which need to be followed. All of them are explained well and can be done in a relatively short time frame. The steps include scoring of the criteria, weighting of the criteria and the final evaluation. There are only six criteria given, making the model easily accessible for the user, but there is a risk that not all aspects might be covered. Therefore, the criteria must be adapted in the model that is to be developed. The fact that the weighting of the criteria always has to be overall 10 might limit the user a little and thus should be done differently in the new model. The result of the model is a number between 1 and 100, but unfortunately no recommendation is given how to do classify the results. For the new model,

the range from 1 to 100 could remain the same, but results must be analyzed and a recommendation should be provided.[63]

The Open Source Maturity Model from Capgemini is not approachable, because the data collection can take up to two and a half years. Although it is a well-structured method, there are too many criteria making it difficult to find information for all of them. Another aspect is that there is a lack of information for the approach the model takes. No further input from this model can be used for the approach of the new model, but some criteria can be adapted as will be shown in Chapter 4.[64]

The OpenBRR model is not established as well as the other three ones. Although it is easy to extend and flexible, the evaluation of the model takes a long time and there is a lack of information for the approach. Furthermore, this model is not widely adopted and seams dead. Therefore, only the following small part of this model can be considered to be used for the new one. The number of criteria regarded in this model is realistic with having 12 criteria. This means the important parts for defining the maturity are covered and it is also achievable for the user to evaluate all criteria. For the new model even more than 12 criteria can be chosen though, but they can build on the specific ones from the OpenBRR model.[65]

Finally, the model QSOS is regarded, which consists of four steps. In general, the approach of this model is structured well and it is also popular, but it is too time-consuming to apply. Since the model is very complex and has many criteria, only one aspect of this model is worth being adapted in the new model. The QSOS model provides the best approach for weighting the criteria with a range of three numbers. Hence, it can be considered to let the new model have a range from 1 to 3, where ascending number resemble an increased importance. The result in form of an absolute score as described in the model from Navica is also found in this model and will be used as described above.[66]

[63] Cf. Russo, B. et al. (2010), p. 305
[64] Cf. ibidem, p. 304
[65] Cf. ibidem, p. 308 et seq.
[66] Cf. Origin, A. (2006), p. 15

4 Developed Maturity Model

4.1 Selected Criteria

The analysis of the Open source maturity models from Navica and Capgemini, the Open business readiness rating and the Qualification and Selection of Open Source Software Model, as well as further research lead to the selection of the below explained criteria for the new maturity model. Appendix 3, page 49, shows all criteria used in the described maturity models. The overlapping criteria between them can be found within, which provided a good and reasonable selection of criteria for the new model. The chosen criteria are highlighted in yellow and explained in the following paragraphs.

The **age** or **longevity** of a FLOSS product measures how long it has been available for use and thus implies a FLOSS project's stability and survival capability.[67]
When scoring this criterion, the following factors have to be considered. FLOSS projects that were initiated only recently might be buggy and entail the risk of being stopped suddenly. The older a project is the lower is this risk. Concerning old or established software, the activity level of the community has to be validated as being still active and the actuality of technologies and methods used for development has to be assured.[68]

The **functionality** criterion is used to determine how well a FLOSS product meets the requirements of the average user.[69]
Attention should be paid to the development and release behavior of the developer community. For example, the method "*Release Early and Often*" enables fast error correction and encourages community members to contribute, as their contribution will be visible soon. On the downside, this can mean that the first releases of this software are defective or incomplete.[70]

---

[67] Cf. van den Berg, K. (2005), p. 13
[68] Cf. Golden, B. (2005), p. 103;
Cf. Duijnhouwer and C. Widdows, Capgemini Open Source Maturity Model, http://www.capgemini.com/technology/opensource/solutions.shtml, 08.2003, http://www.seriouslyopen.org/nuke/html/modules/Downloads/osmm/GB_Expert_Letter_Open_Source_Maturity_, retrieval: 30.03.2005 cit. according to van den Berg, K. (2005), p. 13
[69] Cf. OpenBRR.org (2005), p. 7
[70] Cf. Raymond, E.S. (2005), The Cathedral and the Bazaar. First Monday 3(3), http://www.firstmonday.org/issues/issue3_3/raymond/, retrieval: 30.03.2005 cit. according to van den Berg, K. (2005), p. 16

Concerning a FLOSS product’s **license** the type of license should fit with a company’s intension to use the software. As described in Chapter 2.1, copyleft and copyfree licenses can have a significant effect on the usage of a FLOSS product. If a company wants to use the software and maybe adapt it to its requirements but not sell it, a copyleft license would be acceptable. If the intension is to alter the software and to redistribute it, a copyleft license would render this purpose impossible by making derivative products to FLOSS products as well. In addition to the appropriateness for the usage, it is advisable to choose products with licenses approved by the Open Source Initiative.[71]

Regarding the **usability** of FLOSS products, the operator convenience of the user interface and the ease of use with regard to end-users should be considered. Furthermore, the ease of installation, configuration, deployment and maintenance of the software needs to be considered when assessing a product’s usability.[72]

The **performance** of a FLOSS product needs to be included in the decision-making process in order to consider the software’s resource usage, stability and responsiveness under workload. Thereby, conclusions concerning the product’s reliability can be drawn.[73]

In order to determine the software’s appropriateness for future developments, the **scalability** to large or growing environments and thereby to an increasing amount of data needs to be assessed.[74]

The **quality** of a FLOSS product affects the security, maintainability, reliability and the efficient utilization of the software. Therefore, this important criterion can be evaluated by examining the completeness and flawlessness of the design, the source code and the tests.[75]

Security goes along with quality and considers how well the software handles security issues as well as determining how secure it is in general. The programming needs to be accurate in order to match the code formalities. That applies t FLOSS, but also for proprietary software. Furthermore, it depends on the developers’ attention and input during the build-up phase.

---

[71] Cf. Meeker, H. J. (2008), p. 22 et seq.; Cf. van den Berg, K. (2005), p. 14
[72] Cf. OpenBRR.org (2005), p. 7
[73] Cf. ibidem, p. 8
[74] Cf. ibidem
[75] Cf. ibidem, p. 7

Concerning security, FLOSS has two contradicting sides that must be looked at. One half of users have the opinion that when software is disclosed to everybody a security problem might occur and that unstructured changes might damage the software. The other half argues that the openness of source code improves its quality according to the experienced developers who e.g. solve bugs more quickly. Thus, FLOSS "gives both attackers and defenders great power over system security"[76] what needs to be considered when scoring the security criterion.[77]

There are two different kinds of **support** services for FLOSS products:

- Usage support - the responding to queries concerning the use of the software
- Failure support or maintenance - resolving of difficulties/problems the software has

Both are mostly used in combination in the case where users might not know how to utilize specific software. This is why the FLOSS product's support offerings are so important and need to be analysed.

There are three possibilities of support which may have a different importance to each company. The first one offered is a bug tracker, which is an application where a user can report problems and specific community developers answer and maintain services. The second option is paid support. In this case, the FLOSS can be accessed by users but the professional support from the community needs to be paid. As a third possibility there are special companies (not the community) that offer support for certain FLOSS products. This support is called third party support.[78]

The fact, that there is a paid support available for FLOSS products, especially third party support, shows that these products are taken seriously, which also is an indicator for a high maturity level. In addition FLOSS products host wiki, forums, newsgroups, e-mail listings etc. which as well lead to a high maturity determination.[79]

Three types of **documentation** of a FLOSS product exist: user, developer and maintainer documentations. In user documentation, specific roles need to be described, as for example, administrator or single user. All functionalities are shown and clarified. Tutorials are presented. The documentation can be e.g. viewed on the project´s website. The developer documentation is more important, as the source code is accurately documented therein.

[76] Hoepman, J./Jacobs, B. (2005)
[77] Cf. ibidem
[78] Cf. ibidem
[79] Cf. Golden, B. (2005), p. 124;
Cf. Duijnhouwer and C. Widdows, Capgemini Open Source Maturity Model, http://www.capgemini.com/technology/opensource/solutions.shtml, 08.2003, http://www.seriouslyopen.org/nuke/html/modules/Downloads/osmm/GB_Expert_Letter_Open_Source_Maturity_, retrieval: 30.03.2005 cit. according to van den Berg, K. (2005), p. 15

Information about how to add or change code or comments as well as how to use specific paragraphs has to be mentioned and must be accessible for the public. Closer information about server-based applications, installations and upgrades are considered in maintainer documentation.[80]

Documents are often underestimated and lagging behind the status of applications, since these documents are often prepared after functionality is already implemented. Therefore, the likelihood of vacancies in a documentation is really large (because of this processed method.) Good documentation helps to utilize information in order to be used in the future. Through it, the project information in it makes the software more comprehensible and invokes the learning effect of a member of an organization.[81]

The **community** criterion is one of the most important ones. A community consists of people that use the software and participate in some way as a user, tester or developer. People are a significant resource because it is the community that does most of the testing and that provides quality feedback. The more people are interested in a FLOSS project and the bigger the community, the better the reputation of a project and its acceptance. If the software was not respectable enough, there would not be so many people who care about its progress. The strength of a community can be measured by for example by considering the number of contributors, the frequency and its growth.[82]

The **controllability of the community** is a criterion used to assess the possibility to influence the developer community with regards to future feature development. Thereby, costs for the company-internal adaption of the applied FLOSS can be saved. Furthermore, for applying companies it is important that the applied FLOSS project is kept alive to ensure that the software products as well as the according support remain in force. To ensure that the FLOSS product will meet a company’s future needs, the controllability of the community should be considered in the assessment of an FLOSS product’s maturity.[83]

Integration is a criterion that is essential for software that is being used in association with additional software, adapted to specific needs or personalized by adding functionality in order to meet the organization´s functional requirements.

One integration possibility is modularity. Modular software can be managed easily and can also be individualized by customizing necessary functionalities without touching the core

[80] Cf. Erenkratz, J./Taylor, R.(2003)
[81] Cf. ibidem
[82] Cf. Golden, B. (2005), p. 21
[83] Cf. de Silva, C. (2009)

software. According to the type of license, the modular extension of FLOSS can be possible without giving everything away as FLOSS – add-on software can be sold as commercial software. Adoption also includes opportunity to adapt plugins, e.g. Eclipse.
There are more and more cooperation and collaborations between FLOSS, which is another integration criterion. The usage of standards in FLOSS environments shows a better maturity of software because the combination of products including standards is integrable more easily.[84]

When looking at the criterion **adoption**, the acceptance and implementation of an Open Source product must be looked at. Especially, regarding the community the market and the industry. Public organizations often consider adopting FLOSS for reasons of cost, transparency, citizen oriented and greater effectiveness in communication and distributing services. It can be measured by regarding the acceptance of a FLOSS product by the market and the position it has compared to the market leader.[85]

Trainings about open source services based on proven industry best practices and skills should be provided for every FLOSS project and thus need to be considered in the evaluation of its maturity. Professional service teams for example groups of people within a community can deliver expert knowledge and advice on everything from the basics to advanced techniques for FLOSS and every training engagement can be customized. As there will be more non-technical users, software developers should provide trainings so that the software will be adapted and used. Different trainings need to be offered by the provider in order to attend users, developers and administrators.[86]

Reports provide an overview of a FLOSS product’s current situation to keep customers up to date. This textual work relays information or recounts certain events in a generally presentable manner. It is addressed to a specific audience which is in this case the community, people that use the software and participate in some way. Furthermore a report reveals results of implementations, investigations or inquiries. Appropriate wording is used within the community so everybody understands and is informed about activities or further actions. One of the most common setups for presenting reports is: Introduction, Methods,

---

[84] Cf. Golden, B. (2005), p. 34;
Cf. Duijnhouwer and C. Widdows, Capgemini Open Source Maturity Model, http://www.capgemini.com/technology/opensource/solutions.shtml, 08.2003, http://www.seriouslyopen.org/nuke/html/modules/Downloads/osmm/GB_Expert_Letter_Open_Source_Maturity_, retrieval: 30.03.2005 cit. according to van den Berg, K. (2005), p. 17
[85] Cf. Wang, H./Wang, C. (2001), p. 91 et sqq.
[86] Cf. ibidem, p. 91

Results and Discussion (problem solving). There are differences between a report and documentation. The first one discusses profound patterns and functions whereas documentation demonstrates a fixed status. The more often reports are published, the better users are briefed. In addition to that, the richness of a report's content should be considered in the score of this criterion.[87]

4.2 Weighting Factors for the Calculation of the Overall Maturity Score

After the criteria have been chosen for the model, the next step is to weight them, so that the criteria are tailored to the specific needs of the user. The **weighting** is applied differently in each of the models described in Chapter 3.

The Open Source Maturity Model from Navica delivers a default weighting of the criteria. The model delivers a default list which defines the weighting for each criterion. This enables an easy approach to use the model. In order to adapt the criterion to the individual needs the weighting can be changed, but the sum of all weighting factors always has to be 10. Since, the new model should on the one hand be an easy approach for the user to define the maturity and on the other hand fit to the specific needs, this weighting approach will not be used for the new model.[88]

Another approach for the weighting is described in the OpenBRR model. When 7 out of the 12 criteria are selected these 7 are weighted so that the sum will be 100%. Also, the metrics in each criterion are weighted where the sum has to be 100% as well. This method is rather difficult for the user to apply and also time-consuming. Therefore, it will not be applied in the new model either.[89]

The Open Source Maturity Model from Capgemini defines the weighting of the criteria by discussing it with the customer. This is a good approach for adapting it to the user's need, but unfortunately no details about the weighting process are given. As a result the weighting of this model cannot be used for the new model, either.[90]

Finally, the QSOS model has three categories available for the different relevance of each criterion, which are irrelevant, relevant and critical. This is a great approach to classify the criterion, because it is easy for the user to make the decision for each criterion. Furthermore,

[87] Cf. Bowden, J. (2004), p. 15
[88] Cf. Russo, B. et al. (2010), p. 305
[89] Cf. OpenBRR.org (2005), p. 17
[90] Cf. Russo, B. et al. (2010), p. 304 et seq.; cf. NRC FOSS (2009)

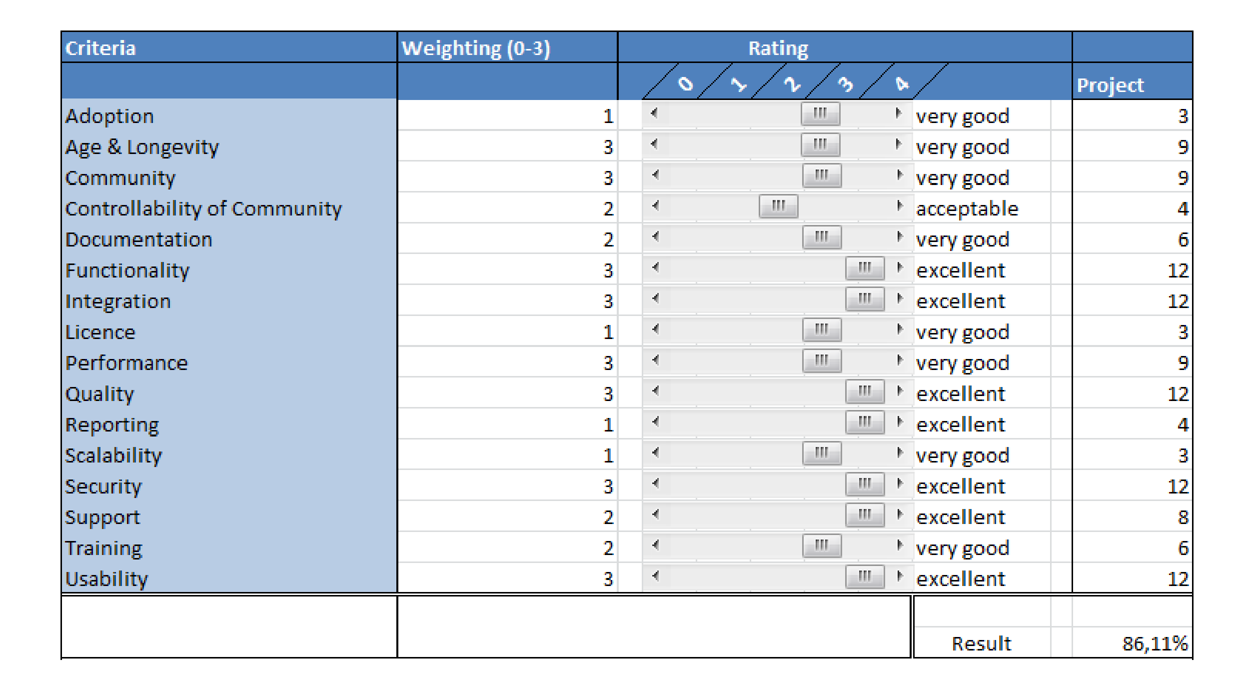
this leads to less result variance.[91] All in all, this is seen as the best approach, because there are clear weighting categories and it is easy to apply. Therefore, for the new model three weighting categories are used from 1 – 3. Additionally, the category 0 is added as a fourth category for the weighting to allow the user to choose only the criteria he/she would like to regard. When 0 is selected this means the according criterion is not selected and therefore not regarded in the maturity model.

After the criteria are weighted the evaluation or so-called rating follows. Therefore, different approaches are found in the four models described in Chapters 3.1 to 3.4. Firstly, the widest range for **scoring** is used in the Open Source Maturity Model from Navica having 10 different categories. While in this case the user has the widest range of possibilities for the evaluation, it is not clearly defined though what each number represents. Therefore, no clear separation of the scores can be identified and they might be seen differently by various users. All in all, the scaling with the numbers from 0 to 10 points is subjective and will not be used.[92]
Secondly, the smallest scoring range used is three in the QSOS model. Hereby, each scoring number is clearly defined (not covered = 0, partially covered = 1 and completely covered = 2) and this will lead to less result variance. The problem in this scoring range is that not all users might find the fitting score, because it is limited to only three categories.[93]
Therefore, the final approach with five categories for the scoring has been chosen. This is used by the two models Open Source Maturity Model from Capgemini and OpenBRR. While in the Open Source Maturity Model from Capgemini 3 indicates a neutral value, numbers below 3 symbolize a negative effect and numbers above 5 represent positive effects.[94] For the OpenBRR model the five scores range from 1 resembling an unacceptable state to 5 as an excellent fulfillment of the criteria. The other scoring numbers stand for the following: 2= poor, 3=acceptable and 4 =very good. Within these five categories the user has an optimum range to evaluate each criteria and each one of them is clearly defined. Therefore, this approach has been identified as the best one and is chosen for the developed model.[95]

[91] Cf. Origin, A. (2006), p. 32
[92] Cf. Russo, B. et al. (2010), p. 306
[93] Cf. Deprez, J./Alexandre, S. (w.y.), p. 8 et seq.
[94] Cf. Russo, B. et al. (2010), p. 304 et seq.; cf. NRC FOSS (2009)
[95] Cf. Russo, B. et al. (2010), p. 308 et seq.

4.3 Introduction of the New Maturity Model

The Excel file containing the developed application for the ***Ready for Business*** maturity model includes three different sheets: one for the standard version of the model, one in which the criteria for evaluation can be changed, and a third one including explanations of each criterion.



Criteria	Weighting (0-3)	Rating	Project
		0 1 2 3 4	Project
Adoption	1	very good	3
Age & Longevity	3	very good	9
Community	3	very good	9
Controllability of Community	2	acceptable	4
Documentation	2	very good	6
Functionality	3	excellent	12
Integration	3	excellent	12
Licence	1	very good	3
Performance	3	very good	9
Quality	3	excellent	12
Reporting	1	excellent	4
Scalability	1	very good	3
Security	3	excellent	12
Support	2	excellent	8
Training	2	very good	6
Usability	3	excellent	12
		Result	86,11%

Fig. 2: Ready For Business Model

As shown by figure 2, the standard version of the Ready for Business model contains the criteria that were described in Chapter 4.1 in the left column. The weighting factors are represented in the column right next to it, whereby users can choose between a onefold, twofold or threefold weighting for each criterion. In the third column, the scroll bars can be used to set the rating for each criterion, whereby users can choose values between one (unacceptable) and five (excellent). The written state of the selected rating is displayed beside the scroll bars, the value for each criterion is shown in the column on the right, and the overall result is displayed below the table.

According to the achieved overall result, a FLOSS product can be categorized as *lacking of basic requirements*, *low*, *moderate* or *high* maturity, as shown in table 7.

Achieved Percentage	0 – 25 %	25 – 50 %	50 – 75 %	75 – 100 %
Maturity Level	Lacking of basic requirements	low	moderate	high

Table 7: Maturity Levels

In the second sheet of the developed solution, users are given the possibility to add or remove criteria in order to customize the criteria selection. The explanations contained in the third sheet tops the solution off, as it enables users to look up an explanation for each criterion within the Ready for Business application.

The possibility to enter values directly into the Excel solution helps users to stay on top of things and to keep track of the impact of the entered values. Hence, this solution contains an easy to use excel application, in which the weighting, scoring and the final result are clearly displayed and thus it simplifies the determination of a FLOSS product's maturity.

The application of this model will be presented by its application to chosen case studies in Chapter 5.2 of this paper.

5 Maturity of Appropriate Case Studies

5.1 Selection of Case Studies

Firefox is a software product from the non-profit organization Mozilla. The open community Mozilla was created in 1998 and since then many people have contributed to the different projects. The first release of the project Firefox was in 2003 with the product Firefox 1.0. Firefox is a web server, which is available to download for free.[96] Its intention was to create "a robust, user-friendly and trustworthy web experience."[97]

VLC media player is one of the VideoLAN project's core applications, which is able to play a wealth of video as well as audio formats and stream content. It also provides a possibility to play and record net streams.[98] Its development started in 1999 and goes on because of its property of being a project under the GNU General Public License.[99] With VLC the user may apply special effects to videos e.g. gradients, waves, motion blur, logos, etc. as well as create their own audio file by recording it.[100] It has been taken as a case study because of its growing popularity, following right up after Windows Media Player and iTunes with the difference and huge benefit of being an open source product. While Windows Media Player and iTunes are restricted to play some files, VLC plays nearly any video file.[101]

webOS is a smartphone and tablet operating system from Hewlett-Packard and it is based on a Linux Kernel. It was initially developed by Palm, which was later purchased by Hewlett-Packard. Palm introduced webOS in January 2009.[102] Several versions of webOS have been designed for several appliances including Pre, Pixi, and Veer phones and the HP TouchPad tablet. This operating system gives to possibility to use multi-touch gestures. Furthermore, it is based on the operating system and user interface developed by Linux. On the 18th of August 2011 the ex-CEO of HP, Leo Apotheker, announced the termination of the development of webOS based devices. Meg Withman, his successor, subsequently **Android** is an operating system for mobile devices like smartphones or tablet computers that is developed and maintained by the Open Handset Alliance.[103] This community is led by a broad consortium of 93 mobile operators, handset manufacturers, semiconductor companies,

[96] Cf. Mozilla Corporation (2012a)
[97] Mozilla Corporation (2004):
[98] Cf. Popov, D. (2005), p. 165
[99] Cf. Bohling, F. (2012)
[100] Cf. Informer Technologies, Inc. (2012)
[101] Cf. Donahue, A. (w.y.)
[102] Cf. Hewlett-Packard Company (2012b)
[103] Cf. Android open source project (w.y.)

software companies and commercialization companies including Vodafone, Dell, Intel, Google, and Accenture.[104] Furthermore, Android is released under the Apache License, which is a copyfree license. Android's large community develops applications in order to extend the functionality of devices. These applications can be downloaded through online stores, whereby 466,651 applications are available for the currently more that 331 million activated android devices.[105] At the end of 2010, Android became the worldwide leading smartphone platform and has reached a global market share of 59 percent in the first quarter of 2012.[106]

5.2 Result of Case Studies

The result of all case studies is shown in the figure 3. In this study it is shown that VLC is the product with the highest maturity of 89.58%. Furthermore, Mozilla Firefox and Android have high maturities with values of 86.11% and 82.64% respectively. WebOS is rated with the lowest maturity of 55.56%. This means that WebOS currently is in the maturity level of being moderate, while all three other FLOSS products are already in the best maturity level which is called a high maturity.

Criteria				
	Mozilla	VLC	WebOS	Android
Adoption	3	4	0	4
Age & Longevity	9	12	3	6
Community	9	9	6	9
Controllability of Community	4	4	0	0
Documentation	6	6	6	6
Functionality	12	12	6	12
Integration	12	12	6	12
Licence	3	3	4	4
Performance	9	12	6	9
Quality	12	12	9	12
Reporting	4	4	2	4
Scalability	3	2	3	4
Security	12	9	9	9
Support	8	8	6	8
Training	6	8	8	8
Usability	12	12	6	12
Result	86,11%	89,58%	55,56%	82,64%

Fig. 3: Rating of Case Studies

[104] Cf. open handset alliance (w.y.)
[105] Cf. AppTornado GmbH (2012); cf. Vocus (2012)
[106] Cf. Canalys (2012); cf. Vocus (2012)

As an example, the evaluation of the maturity of the VLC media player is described in the subsequent chapter. Further information about the determination of the maturity of Mozilla Firefox, WebOS and Android can be found in appendix 4, page 50, appendix 5, page 52, and appendix 6, page 54.

5.3 Maturity of the Example

As an example for a detailed evaluation within this paper VLC Media Player has been chosen. Starting with the first criteria, **age and longevity** received a rating of 5**.** VLC Media Player exists since 1999[107]. Statistically, the international VLC community which is open to everyone interested has 304,003 total posts, 79,046 topics and 69,347 total members so far[108] while the German community currently has 571 topics and 1035 contributions which makes an average of 1.08 contributions a day. Furthermore, there is a developer community where developers can develop VLC patches and send it before it will get approved and appear in the VLC-commits list, where all the approved patches are being listed. Patches will be considered in the new versions. As a specific example, the latter community shows 246 contributions in June and 542 in May.[109] Also to consider for longevity is the actuality of technologies and methods used for development. Programmers can help contribute by integrating the project's new features, features improvement, bug tracking, code cleaning, porting to a new platform either in C, C++, or ASM.[110] The criteria that follow right along will be described in notes providing a better overview for the reader.

Functionality (Overall Rating: 5)

- Development and release behavior of community:
    - Developers develop patches for VLC with high frequency → 246 contributions in June, 542 in May[111]
    - Have a forum with bug reports→ About 10 bugs/missing features reported a day
        - not excluding duplicates, some are invalid→ vague approximation: about 1/3 of them are actual bugs
        - have different priority, some are just typing errors etc., others have major priority
        - Have a timeline with milestones showing the progress on bugs[112]

[107] Cf. Bohling, F. (2012)
[108] Cf. phpBB Group (2007)
[109] Cf. Kempf, J.-B. (2012a)
[110] Cf. Kempf, J.-B. (2012b)
[111] Cf. Kempf, J.-B. (2012a)

- Meeting requirements of average user:
  - User rating (from users that downloaded VLC): 4 out of 5 stars (3276 votes)[113]
  - User rating (from users that downloaded VLC): 98% thumbs up, 2% thumbs down (179.657)[114]
- Frequency of releases:
  - Ø 7,4 releases from 2005 to 2011[115]

License (Overall Rating: 4)

- Software patents licenses do not apply on VideoLAN[116]
- Copyleft→it may be distributed under the terms of this General Public License[117] →no redistribution

Usability (Overall Rating: 5)

- Operator convenience of the user interface, ease of use, ease of installation, configuration deployment, maintenance:
  - As shown above these points are included in the user ratings
    - User rating (from users that downloaded VLC): 4 out of 5 stars (3276 votes)[118]
    - User rating (from users that downloaded VLC): 98% thumbs up, 2% thumbs down (179.657)[119]

Performance (Overall Rating: 5, see appendix 7, page 56)

- Very good minimum launch time (2nd best)
- Lowest maximum launch time
- Very good CPU utilization (pretty low)
- Average minimum RAM use
- Medium-good maximum RAM use[120]

Scalability (Overall Rating: 3)

- No information about scalability found

---

112 Cf. Edgewall Software (2012)
113 Cf. CNET Staff (2009)
114 Cf. CHIP Xonio Online GmbH (2012)
115 Cf. FileHippo.com (2012)
116 Cf. Kempf, J.-B. (2012c)
117 Cf. Free Software Foundation, Inc (2012)
118 Cf. CNET Staff (2009)
119 Cf. CHIP Xonio Online GmbH (2012)
120 Cf. Williams, M. (2011)

Quality (Overall Rating: 5)

- **Completeness, flawlessness (design, source code):**
    - With VLC you can manually influence sound and appearance[121]
    - User Interface: ★★★★½
      Features: ★★★★☆
      Ease of use: ★★★★☆
      Total Rating: Good[122]

Security (Overall Rating: 4)

- 2 security flaws identified in 2011 which had a highly critical nature[123]
- No further serious vulnerabilities found in the new version so far, at least cannot be found on the world wide web

Support (Overall Rating: 5)

- **Usage support, Failure support (resolving of difficulties), Bug tracker/ paid support (professional support from community is paid), external company offering support:**
    - Troubleshooting guide exists, user can search in VLC forum, find issue by reading the wiki
    - User may contribute to forum
    - Videos for training
    - VLC live-chat[124]
    - Support within the tool

Documentation (Overall Rating: 5)

- User, developer and maintainer documentation
    - **User documentation:** roles are defined, tutorials presented, doc. is on website
        - Tutorial for Installing VLC, other basic features
        - Roles defined: roles for contribution are clearly defined[125]
        - Doc on website: yes http://wiki.videolan.org/Documentation
    - **Developer documentation:**
        - accurately documented code: yes
        - info how to add/amend code: Code conventions accessible to everyone[126]
    - **Maintainer documentation:**
        - Lack of information on maintenance[127]

---

[121] Cf. Computer Videos (2012)
[122] Cf. VLC Media Player source code Pro! (2012)
[123] Cf. Constantin, L. (2011)
[124] Cf. Kempf, J.-B. (2012c)
[125] Cf. Kempf, J.-B. (2012b)
[126] Cf. ibidem
[127] Cf. ibidem

Community (Overall Rating: 4)

- Cannot say how many developers exist
- Forum exists, where bugs are reported/questions asked by users, answers are given to about 50% of them[128]

Controllability of the community (Overall Rating: 3)

- Community needs to fix bugs and provide new patches
- →BUT depends on the company→ cannot really be evaluated

Integration (Overall Rating: 5)

- Linux, Windows, Mac OS X, BeOS, BSD, Solaris, Familiar Linux, Yopy/Linupy, QNX and many more...[129]

Adoption (Overall Rating: 5)

- Acceptance (in community, in market, in industry):
  - Version 2.0.1 has been downloaded 146928170 times already[130]
  - Is market leader in segment (open source media players)[131]

Trainings (Overall Rating: 5)

- External trainings exist[132]
- Several training videos
- VLC Community calls up to create trainings in the VLC Wiki[133]

Reports (Overall Rating: 5)

- Bug reporting[134]
- Besides that no information on further reports was found

[128] Cf. WoltLab GmbH (2012)
[129] Cf. Kempf, J.-B. (2012b)
[130] Cf. ibidem
[131] Cf. Sharewareconnetion (w.y.)
[132] Cf. FLIR Systems, Inc. (2012)
[133] Cf. Kempf, J.-B. (2012)
[134] Cf. ibidem

6 Critical Reflection on the Maturity Model Introduced

As models are used to observe the reality in an abstracted and simplified way, according to their kind, they present some limits due to the simplification.[135] Therefore, the Ready for Business model will be critically examined in the following.

In order to be able to apply a maturity model, it needs to be understood at first. Therefore, a comprehensible **description** of the model needs to be available. For the Ready for Business model this paper contains a detailed description and the application of the Excel solution is described in a user manual. As a next step, the **data collection** can raise problems concerning the availability and accessibility of information about FLOSS products. In contrast to the Open Source Maturity Model from Capgemini, the Ready for Business model does not prescribe a timeframe for the collection of data, wherefore this can be defined as desired.[136] Despite the detailed description of the criteria used in the developed model, the **interpretation** of the collected data might be difficult. Different users of the maturity model might evaluate with individual views on a FLOSS product and hence the evaluation is influenced by a subjective aspect. This fact cannot be avoided, but has to be kept in mind when applying the model. In order to evaluate the data, the user of the model has to know the requirements on the desired FLOSS product. These requirements provide the basis for the evaluation of FLOSS and thus for the assessment of data. In order to give different evaluation criteria a different **weighting**, the weighting of the QSOS model is adopted. Users can decide whether a criterion shall be included or not and a differentiation between irrelevant, relevant and critical criteria is enabled by the allocation of a onefold, twofold or threefold weighting. On the one hand, this weighting is easy to use and leads to less result variance. On the other hand, this classification is not too finely graduated.[137] Concerning the subsequent **scoring** of the criteria, values between zero and four can be chosen. At this, the values zero and four are easy to understand, as they imply the worst and best scoring values. The graduation between these excesses might be difficult. However, this difficulty is not as serious as with the Open Source Maturity Model from Navica, which provides ten scoring categories, and according to Russo, five categories provide an optimum range to evaluate each criterion.[138] Mettler and Rohner recommend the **configurability** of a maturity model.[139] Regarding the Ready for Business model, users can decide whether to include the suggested criteria or not. In addition, further criteria can be added in order to customize the model according to the individual needs of applying users.

---

[135] Cf. Leitenberger, B. (w.y.)
[136] Cf. Russo, B. et al. (2010), p. 304
[137] Cf. Origin, A. (2006), p. 32
[138] Cf. Russo, B. et al. (2010), p. 308 et seq.
[139] Cf. Mettler, T./Rohner, P. (2009)

As the development of the Ready for Business model is based on the analysis and comparison of different maturity models and aims to be implementable, it has a profound basis. Despite the configurability of the criterion, the weighting and scoring values cannot be adapted, wherefore this model cannot be generalized as fitting for each applicant. Nevertheless, the ease of application and the availability of the Excel solution, which simplifies the calculation of a FLOSS product’s maturity, argue for the Ready for Business model.

7 Conclusion

The following chapter shows the value of this paper and provides a summary with the most important results from the research.

7.1 Summary

As globalization continues to grow as an issue and the variety of FLOSS products gets wider and more transparent, it becomes even more essential for companies to find the right software (package) that suits their objectives in order to gain an edge over competitors. With the help of maturity models an evaluation of Open Source Software can be processed by using different criteria, for example usability or functionality.

Recapitulating, this paper analyzed four main maturity models with the aim to gain an overview of the methodology and features used. long processing times, less comprehensive results or a lack of user guidance were some of the results concerning the usability of maturity models. Besides this, software is no longer solely used by computer developers; the number of nontechnical users is growing constantly and so is the need of an appropriate, end-user friendly measurement model. Due to the analysis and the inferred findings a new user-friendly, quickly processable maturity model, named **Ready for Business** providing a final expressive result was developed in form of an Excel tool. Some criteria as well as methodologies of the initially processed maturity models were taken over to the new tool. Thus, e.g. criteria that was found in the majority of the models was evaluated as criteria most consider rational for evaluating a FLOSS. Through the newly developed model, the determination of the maturity of a FLOSS product will now be much easier and convincing. In order to proof this fact and show how the tool works, it was applied to appropriate case studies. Four products were picked. Some of them are famous and often applied and others unknown. Applying the tool for the case study was a perfect chance to test the Ready for Business model and came to the result that all of the characteristics chosen are justified since all aspects of the software are enlightened making the user able to say if the software fits their needs or not. Also, the model is very easy to use for a wider variety of possible users. The target group, which is mainly companies of whatever size, usually knows how to use the excel tool and it is very easy comprehensible. The basis for calculation is made clear to the user and the format provides a constant overview of all aspects. All in all, the analysis carried out at the beginning, the development of a new maturity model and finally application of this model in a case study for the paper proceeded well, was finished in time and delivered

a comprehensive result in form of an Excel tool for the usage of determining a maturity of FLOSS products.

7.2 Outlook

The new tool **Ready For Business** can still undertake further research and developments since there is always room for improvement especially as technology changes. Currently, users determine the maturity on a subjective base. Objective ratings are more meaningful and universal and will lead to an improvement of the tool. To achieve this, the analyzed criteria have to be defined more precisely in order to be representative and to show exactly how the criteria can be identified when determining the maturity.
This paper includes a case study with four different software products. More tests e.g. with software products from the same industry should be conducted to establish comparability.
Furthermore, a website for the new developed model may be launched in order to gain improvement suggestions by a circle of interested people. Also, for this kind of model a community can be build up, which might implement further ideas and provide a whitepaper.
In addition, it would be nice for the authors of this paper to know the number of adoptions of this model and whether it worked out fine.
Moreover, it may be possible to alter the model by supporting other programs besides Excel so as to define the maturity of a software.
These mentioned improvements will lead to a more flexible and even more user-friendly tool.
With the paper a good starting point has been laid out for the usage of a new maturity model and will help customers in taking the right decision regarding the demand for open software.

Appendix

List of appendices

Appendix 1: CMMI Maturity Levels and Process Areas 41
Appendix 2: QSOS - Criteria for User Risk 42
Appendix 3: Selection of Criteria for the Ready For Business model 49
Appendix 4: Case Study Mozilla 50
Appendix 5: Case Study WebOS...52
Appendix 6: Case Study Android...54
Appendix 7: VLC Media Player Performance Compared 56

Appendix 1: CMMI Maturity Levels and Process Areas[140]

Maturity Level	*Name*	*Focus*	*Process Areas*
1	Initial		None
2	Managed	Basic Project Management	Requirements Management
			Project Planning
			Project Monitoring and Control
			Supplier Agreement Management
			Measurement and Analysis
			Process and Product Quality Assurance
			Configuration Management
3	Defined	Process Standardization	Requirements Development
			Technical Solution
			Product Integration
			Verification
			Validation
			Organizational Process Focus
			Organizational Process Definition
			Organizational Training
			Integrated Project Management, Specific Goals 1 and 2
			Integrated Project Management, Specific Goals 3 and 4
			Risk Management
			Decision Analysis and Resolution
			Organizational Environment for Integration
			Integrated Teaming
			Integrated Supplier Management
4	Quantitatively Managed	Quantitative Management	Organizational Process Performance
			Quantitative Project Management
5	Optimizing	Continuous Process Improvement	Organizational Innovation and Deployment
			Causal Analysis and Resolution

[140] Contained in: Kenett, R.S./Baker, E. (2010)

Appendix 2: QSOS - Criteria for User Risk[141]

Intrinsic durability		Score		
		0	**1**	**2**
Maturity	**Age**	For instance less than 3 months	For instance between 3 months and 3 years	For instance more than 3 years
	Stability	Unstable software with numerous releases or patches generating side effects	Stabilized production release existing but old. Difficulties to stabilize development releases	Stabilized software. Releases provide bug fixes corrections but mainly new functionalities
	History, known problems	Software knows several problems which can be prohibitive	No known major problem or crisis	History of good management of crisis situations
	Fork probability, source of Forking	Software is very likely to be forked in the future	Software comes from a fork but has very few chances of being forked in the future	Software has very little chance of being forked. It does not come from a fork either

Intrinsic durability		Score		
		0	**1**	**2**
Adoption	**Popularity (related to: general public, niche, ...)**	Very few users identified	Detectable use on Internet (sourceforge, freshmeat, google, ...)	Numerous users, numerous references
	References	None	Few references, non critical usages	Often implemented for critical applications
	Contributing community	No community or without real activity (forum, mailing list,...)	Existing community with a notable activity	Strong community: big activity on forums, numerous contributors and advocates
	Books	No book about the software	Less than 5 books about the software are available	More than 5 books about software are available, in several languages

[141] Contained in: Origin, A. (2006), p. 16 et sqq.

Intrinsic durability		Score		
		0	**1**	**2**
Development leadership	**Leading team**	1 to 2 individuals involved, not clearly identified	Between 2 and 5 independent people	More than 5 people
	Management style	Complete dictatorship	Enlightened despotism	Council of architects with identified leader (e.g: ASF, ...)

Intrinsic durability		Score		
		0	**1**	**2**
Activity	**Developers, identification, turnover**	Less than 3 developers, not clearly identified	Between 4 and 7 developers, or more unidentified developers with important turnover	More than 7 developers clearly identified, very stable team
	Activity on bugs	Slow reactivity in forum or on mailing list, or nothing regarding bug fixes in release notes	Detectable activity but without process clearly exposed, long reaction/resolution time	Strong reactivity based on roles and tasks assignment
	Activity on functionalities	No or few new functionalities	Evolution of the product driven by the core team or by user's request without any clearly explained process	Tool(s) to manage feature requests, strong interaction with roadmap
	Activity on releases	Very weak activity on both production and development releases	Activity on production and development releases. Frequent minor releases (bug fixes)	Important activity with frequent minor releases (bugs fixes) and planned major releases relating to the roadmap forcast

Intrinsic durability		Score		
		0	1	2
Independence of developments	**Independence of developments**	Developments realized at 100% by employees of a single company	60% maximum	20% maximum

Industrialised solution		Score		
		0	1	2
Services	**Training**	No offer of training identified	Offer exists but is restricted geographically and to one language or is provided by a single contractor	Rich offers provided by several contractors, in several languages and split into modules of gradual levels
	Support	No offer of support except via public forums and mailing lists	Offer exists but is provided by a single contractor without strong commitment quality of services	Multiple service providers with strong commitment (e.g: guaranteed resolution time)
	Consulting	No offer of consulting service	Offer exists but is restricted geographically and to one language or is provided by a single contractor	Consulting services provided by different contractors in several languages

Industrialised solution		Score		
		0	1	2
Documentation	**Documentation**	No user documentation	Documentation exists but shifted in time, is restricted to one language or is poorly detailed	Documentation always up to date, translated and possibly adapted to different target readers (end user, sysadmin, manager, ?)

Industrialised solution		Score		
		0	1	2
Quality Assurance	**Quality Assurance**	No QA process	Identifies QA process but not much formalized and with no tool	Automatic testing process included in code's life-cycle with publication of results
	Tools	No bug or feature request management tool	Standard tools provided (for instance by a hosting forge) but poorly used	Very active use of tools for roles/tasks allocation and progess monitoring

Industrialised solution		Score		
		0	1	2
Packaging	**Source**	Software can't be installed from source without a lot of work	Installation from source is limited and depends on very strict conditions (OS, arch, lib, ...)	Installation from source is easy
	Debian	The software is not packaged for Debian	A Debian package exists but it has important issues or it doesn't have official support	The software is packaged in the distribution
	FreeBSD	The software is not packaged for FreeBSD	A port exists but it has important issues or it doesn't have official support	A official port exists in FreeBSD
	HP-UX	The software is not packaged for HP-UX	A package exists but it has important issues or it doesn't have official support	A tested package is provided for HP-UX
	MacOSX	The software is not packaged for MacOSX	A package exists but it has important issues or it doesn't have official support	The software is packaged in the distribution
	Mandriva	The software is not packaged for Mandriva	A package exists but it has important issues or it doesn't have official support	The software is packaged in the distribution

Industrialised solution		Score		
		0	**1**	**2**
Packaging	**NetBSD**	The software is not packaged for NetBSD	A port exists but it has important issues or it doesn't have official support	A official port exists in NetBSD
	OpenBSD	The software is not packaged for OpenBSD	A port exists but it has important issues or it doesn't have official support	A official port exists in OpenBSD
	RedHat/Fedora	The software is not packaged for RedHat/Fedora	A package exists but it has important issues or it doesn't have official support	The software is packaged in the distribution
	Solaris	The software is not packaged for Solaris	A package exists but it has important issues or it doesn't have official support (e.g: SunFreeware.com)	The software is supported by Sun for Solaris
	SuSE	The software is not packaged for SuSE	A package exists but it has important issues or it doesn't have official support	The software is packaged in the distribution
	Windows	The project can't be installed on Windows	A package exists but is limited or has important issues or covers only specific Windows releases (e.g: Windows2000 and WindowsXP)	Windows is fully supported and a package is provided

Technical adaptability		Score		
		0	**1**	**2**
Modularity	**Modularity**	Monolithic software	Presence of high level modules allowing a first level of software adaptation	Modular conception, allowing easy adaptation of the software by selecting modules or even developing new ones

Technical adaptability		**Score**		
		0	**1**	**2**
By-products	**Code modification**	Everything by hand	Recompilation possible but complex without any tools or documentation	Recompilation with tools (e.g: make, ANT, ...) and documention provided
	Code extension	Any modification requires code recompilation	Architecture designed for static extension but requires recompilation	Principle of plug-in, architecture designed for dynamic extension without recompilation

Strategy		**Score**		
		0	**1**	**2**
License	**Permissiveness (to be weighted only if user wants to become owner of code)**	Very strict license, like GPL	Moderate permissive license located between both extremes (GPL and BSD), dual-licensing depending on the type of user (person, company, ...) or their activities	Very permissive like BSD or Apache licenses
	Protection against proprietary forks	Very permissive like BSD or Apache licenses	Moderate permissive license located between both extremes (GPL and BSD), dual-licensing depending on the type of user (person, company, ...) or their activities	Very strict license, like GPL

Strategy		**Score**		
		0	**1**	**2**
Copyright owners	**Copyright owners**	Rights held by a few individuals or entities, making it easier to change the license	Rights held by numerous individuals owning the code in a homogeneous way, making relicensing very difficult	Rights held by a legal entity in whom the community trusts (e.g. FSF or ASF)

Strategy		Score		
		0	1	2
Modification of source code	**Modification of source code**	No practical way to propose code modifications	Tools provided to access and modify code (like CVS or SVN) but not really used to develop the software	The code modification process is well defined, exposed and respected, based on roles assignment

Strategy		Score		
		0	1	2
Roadmap	**Roadmap**	No published roadmap	Existing roadmap without planning	Versionned roadmap, with planning and measure of delays

Strategy		Score		
		0	1	2
Sponsor	**Sponsor**	Software has no sponsor, the core team is not paid	Software has an unique sponsor who might determine its strategy	Software is sponsored by industry

Strategy		Score		
		0	1	2
Strategical independence	**Strategical independence**	No detectable strategy or strong dependency on one unique actor (person, company, sponsor, ...)	Strategical vision shared with several other free and open source projects but without strong commitment from copyrights owners	Strong independence of the core team, legal entity holding rights, strong involvement in the standardization process

Appendix 3: Selection of Criteria for the Ready For Business model

	BRR	QSOS	OSMM Capgemini	OSMM Navica
Acceptance			x	
Activity on releases		x		
Administration			x	
Adoption	x	x		
Advice			x	
Age		x	x	
Architecture	x			
Community	x			
Documentation	x	x		x
Functionality	x			
History known problems		x		
Implementation			x	
Influencability on community				
Integration		x	x	x
Interfacing			x	
Performance	x		x	
Platform Independence			x	
Popularity		x		
Product			x	
Product Software				x
Professional Service				x
Professionalism	x			
Proven technology			x	
Quality	x			
Reliability			x	
Reporting			x	
Scalability	x			
Security	x		x	
Staffing			x	
Support	x	x	x	x
Training		x	x	x
Usability	x		x	
Vendor Independence			x	

Appendix 4: Case Study Mozilla

Criterion	Data
1. Adoption	2nd most used web browser currently after Chrome, also rated 2nd best after Chrome[142]; 220million people customer base[143]
2. Age	Initial release November 9th 2004 → 8 years[144]
3. Community	Core of 400 developers, more than 30% code from volunteers; all in all community of millions of members[145], deliver available chat[146]
4. Documentatio n	Online documentation available for users and developers[147]
5. Functionality	13 mature releases beginning with Firefox 1.0[148]; rapid release process: new version out every 6 weeks[149], many features rated 8.8 out of 10 (organized and intuitive, powerful and useful)[150]
6. Controllability on Community	Hundreds of developers for "add-ons", they create extensions to customize Firefox[151]
7. Integration	Works on Windows 7, Windows Vista, Windows XP, Mac OS[152]; many add-ons available
8. License	MPL(Mozilla Public License)[153] – copyleft [154]
9. Performance	Startup time 6.3sec (3rd place out of 10 web browsers), second out of 10 in navigation time after IE 5.7sec[155]
10. Quality	10,000 people test "nightly" development releases from the community[156], ease of use rated 8.8 out of 10[157]
11. Reports	Bugzilla, Mozilla Project bug and also feature tracking system; 20.000ppl involved[158]
12. Scalability	Many add-ons available, high compatibility
13. Security	integrated private browsing; easy to enable and disable the private mode, protects against viruses, spyware, malware, phishing sites, pop-ups,

[142] Cf. TechMediaNetwork (2012)
[143] Cf. Hyde, A. (2012)
[144] Cf. Mozilla Corporation (2012b)
[145] Cf. Hyde, A. (2012)
[146] Cf. TechMediaNetwork (2012)
[147] Cf. Mozilla Developer Network (2012b)
[148] Cf. Mozilla Corporation (2012c)
[149] Cf. Yam, M. (2012)
[150] Cf. TechMediaNetwork (2012)
[151] Cf. Hyde, A. (2012)
[152] Cf. TechMediaNetwork (2012)
[153] Cf. Mozilla Developer Network (2012a)
[154] Cf. ibidem
[155] Cf. TechMediaNetwork (2012)
[156] Cf. Hyde, A. (2012)
[157] Cf. TechMediaNetwork (2012)
[158] Cf. Hyde, A. (2012)

	customized security settings, automatic updates, easy to clear personal information, includes e.g. history, cookies, passwords and web-form entries; popup blocker[159]
14. Support	Homepage with lots of information including FAQs, a knowledgebase and a tutorial, knowledgebase is searchable, standard help as well as personalized help in form of a live chat with the Firefox community, direct support via email[160] (only exception: no telephone support) [161]
15. Trainings	User guide online for Firefox, YouTube trainings (e.g. how to use it, training on add-ons)[162]
16. Usability	Tabbed browsing, simple installation, synchronization feature, overall performance rated 6/6 by 52%, 5/6 by 33% and 4/6 by 10% of the participants[163] automatic session restore, download manager is easy to use, though separate address bar and a search bar (instead of one)[164]

[159] Cf. TechMediaNetwork (2012)
[160] Cf. ibidem
[161] Cf. Mozilla Corporation (2012d)
[162] Cf. w.a. (2005b)
[163] Cf. Cabello, P. (2007)
[164] Cf. TechMediaNetwork (2012)

Appendix 5: Case Study WebOS

Criterion	Data
1. Adoption	Not many people use webOS as a operating system[165]
2. Age	Initial release June 2009; FLOSS product since December 2011[166]
3. Community	Performance-based selection of developer, tester etc.[167]
4. Documentation	The Open webOS project website hosts a wiki, a source code repository, a mailing list, and a bug tracking system.[168] Extensive documentation[169]
5. Functionality	38 releases[170] since 2009[171] (Multitasking[172], drag-and-drop organization[173], quick navigation) ➔ based on common Web technology WebKit (easier to create additional software)[174] The webOS App Catalog currently has a few thousand official apps available for download.[175]
6. Controllability on Community	- Yet only applied by HP (dependend)[176] - few programmers who understand WebKit →Influence not high[177]
7. Integration	Integration available to native applications[178]
8. License	Open webOS is available under the Apache License, Version 2.0.[179]
9. Performance	Phone is too slow, phone spontaneously restarted itself or freezing up, launch time extremely long. limited storage used[180]
10. Quality	Jobs are offered.[181]
11. Reports	Short reports are provided with main improvements.
12. Scalability	Scalable operating environment – build up apps more rapidly
13. Security	Provides both security guarantees and an interface for authenticating the identity of principals.[182] System policy components run in the background.[183]

165 Cf. Hyde, A. (2012)
166 Cf. Freebase, Metaweb (2012)
167 Cf. Mobspot (2010); cf. appSpotlight (2012); cf. webOS Mobile Nation (2012); cf. Hewlett-Packard Company (2012c)
168 Cf. ibidem
169 Cf. Hewlett-Packard Company (2012d)
170 Cf. Palm (2012)
171 Ct. Freebase, Metaweb (2012)
172 Cf. Carr, Austin (2011)
173 Cf. Hewlett-Packard Company (2012a)
174 Cf. Carr, Austin (2011)
175 Cf. Chen, B. (2012)
176 Cf. McCann, J. (2012)
177 Cf. Webinos (2012)
178 Cf. Allen, M. (2009), p. 201
179 Cf. Hewlett-Packard Company (2012c)
180 Cf. Chen, B. (2012)
181 Cf. Hewlett-Packard Company (2012c)

14. Support	Community[184] Chat Support, Support Portal[185]
15. Trainings	Tutorials[186]
16. Usability	Aim: separates the rendering process from the user interface. This architectural approach delivers smooth scrolling and a responsive user experience[187] webOS is less customizable - you can choose what icons appear in the Launcher and change the wallpaper, but there are no folders for easy app-grouping[188]

[182] Cf. Hewlett-Packard Company (2012a)
[183] Cf. McCann, J. (2012); cf. Palm webOS Security (w.j.)
[184] Cf. Mobspot (2010)
[185] Cf. Palm webOS Security (w.j.)
[186] Cf. Hewlett-Packard Company (2012d); cf. Hewlett-Packard Company (2012a)
[187] Cf. ibidem
[188] Cf. Carr, A. (2011)

Appendix 6: Case Study Android

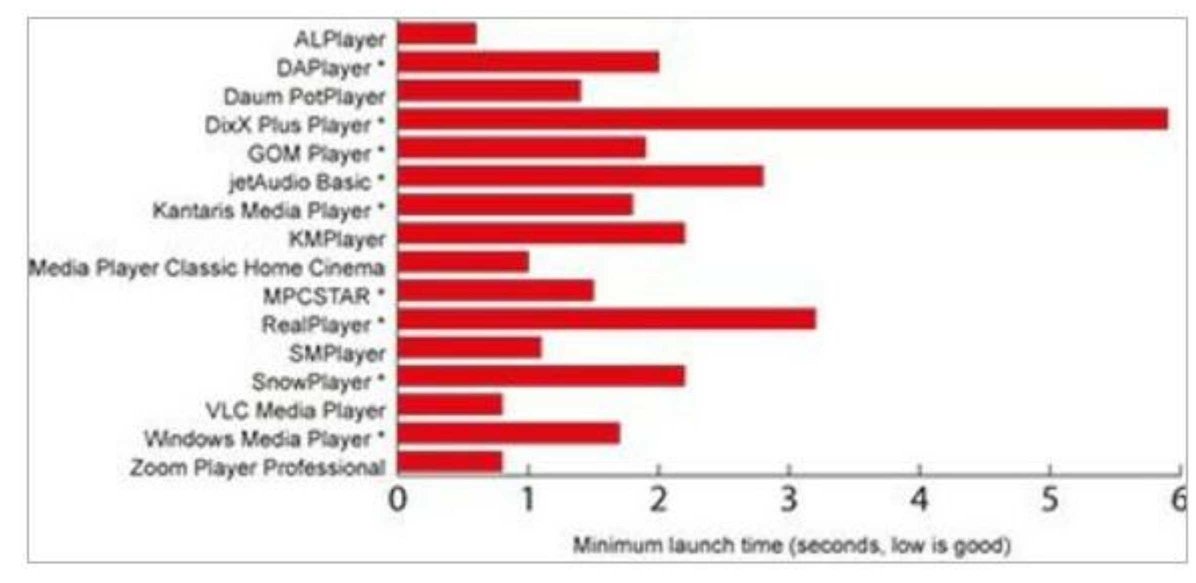
Criterion	Data
1. Adoption	Android is the most used operating system for mobile devices and has a global market share of 59 percent[189]
2. Age	The Open Handset Alliance joined forces and announced the development of Android in 2007 → 5 years[190]
3. Community	The Open Handset Alliance community is led by a broad consortium of 93 mobile operators, handset manufacturers, semiconductor companies, software companies and commercialization companies including for example Vodafone, Dell, Intel, Google, and Accenture. In addition, numerous further participants contribute to the productivity of the community (466651 applications are available).[191]
4. Documentation	A lot of articles are provided online including source code examples and a user guide is provided including user documentation.[192]
5. Functionality	Android provides an operating system, a middleware, a user-friendly interface as well as applications and is the market leader with regards to operating system for mobile devices.[193] Since 2005, 466651 applications were developed.[194]
6. Controllability on Community	The chance to influence the community seems to below, as 93 companies are leading the Open Handset Alliance.[195]
7. Integration	Android can be integrated with Windows via USB and with Microsoft Exchange Sync. Different applications exist that enable the view of common data formats like pdf and the common programming language Java is used to develop Android.[196]
8. License	For the most part, Android is licensed under the Apache Software License - copyfree[197]
9. Performance	Android applications run with limited storage and computing power, and constrained battery life, wherefore it is efficient.[198]
10. Quality	100 percent automated black-box testing for Android is provided by a test management and test automation solution. It covers functional testing, performance testing, stress testing, regression testing, and sanity/smoke Testing.[199] Only 28 percent of the current 466651 applications have a low quality.[200]

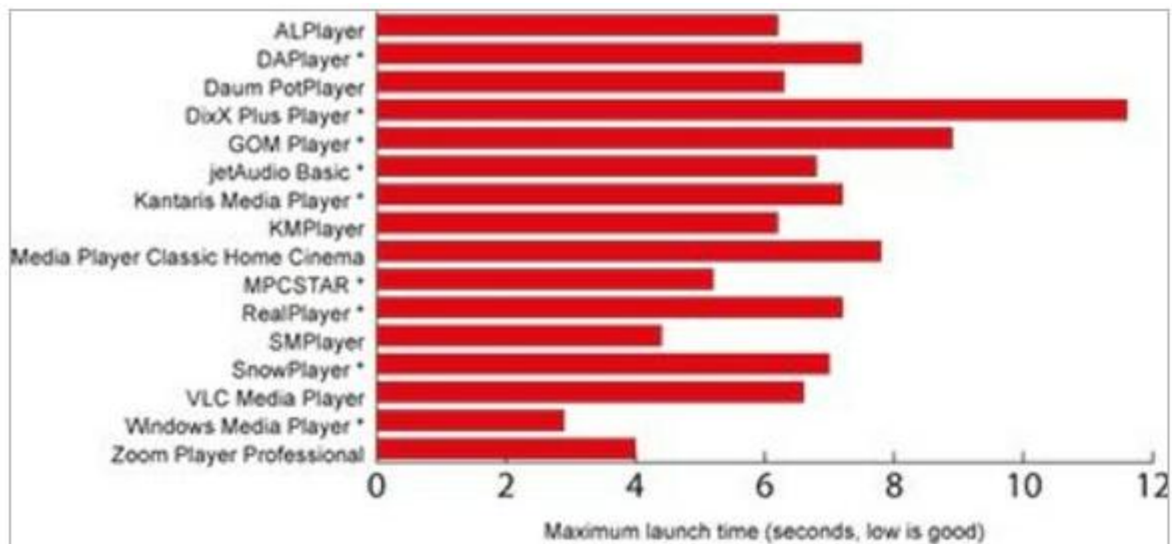
[189] Cf. Canalys (2012); cf. Vocus (2012)
[190] Cf. open handset alliance (2007)
[191] Cf. open handset alliance (w.y.); cf. AppTornado GmbH (2012)
[192] Cf. Android Developers (2012); cf. Google (2011)
[193] Cf. open handset alliance (2007)
[194] Cf. AppTornado GmbH (2012)
[195] Cf. open handset alliance (w.y.)
[196] Cf. Mock, U. (2008)
[197] Cf. Android open source Project (w.y.)
[198] Cf. ibidem
[199] Cf. eInfochips (2011)
[200] Cf. AppTornado GmbH (2012)

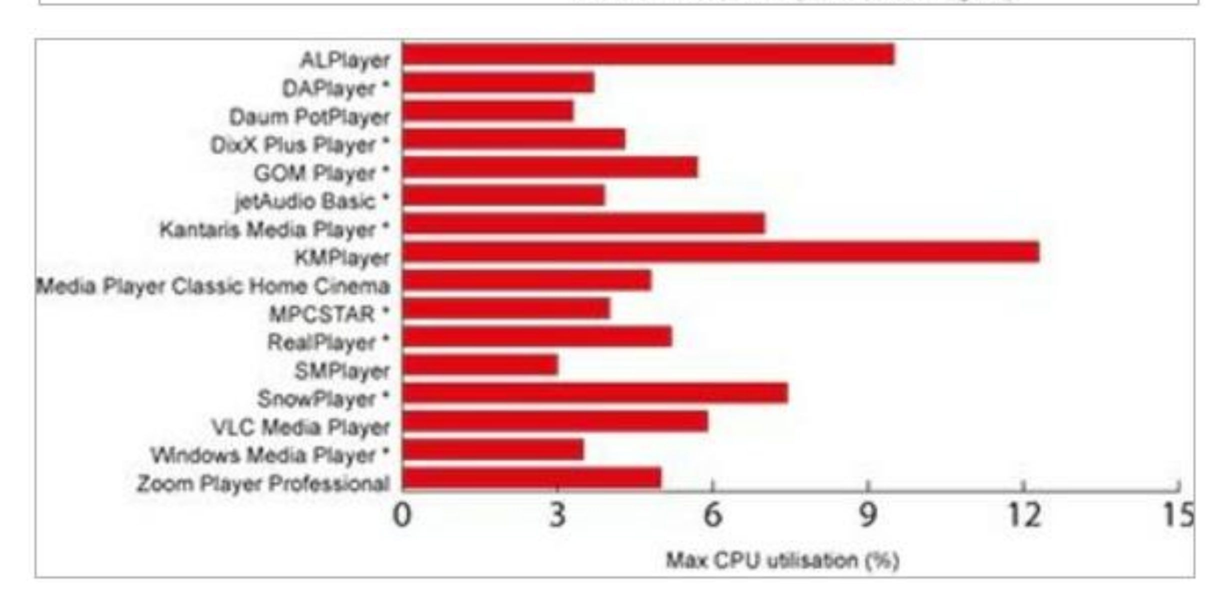
11. Reports	Application reports, error reports[201]
12. Scalability	Many applications available, high compatibility
13. Security	13000 malwares existed for Android in 2011, but anti-virus programs are on the market as well.[202] Android provides developers with a broad range of security requirements that can be used to design secure applications.[203]
14. Support	An Android support forum exists as well as a homepage with lots of information and FAQs. In addition, fee-based support is available as well.[204]
15. Trainings	A collection of classes is provided in order to help contributors to build applications by using best practices. Additionally, vendors provide training courses.[205]
16. Usability	Android facilitates intuitive handling, provides a solid and consistent platform, and applications have a consistent look and feel. For its developers, guidelines are provided in order to improve the usability of developed applications.[206]

[201] Cf. Foulke, D. (2008); cf. Bray, T. (2010)
[202] Cf. Mock, U. (2008)
[203] Cf. Android Developers (2012)
[204] Cf. Brand Solutions (2011); cf. AdroidExpert (w.y.)
[205] Cf. Android Developers (2012)
[206] Cf. Polkamp, C. (2011); cf. Lehtimaki, J. (2012)

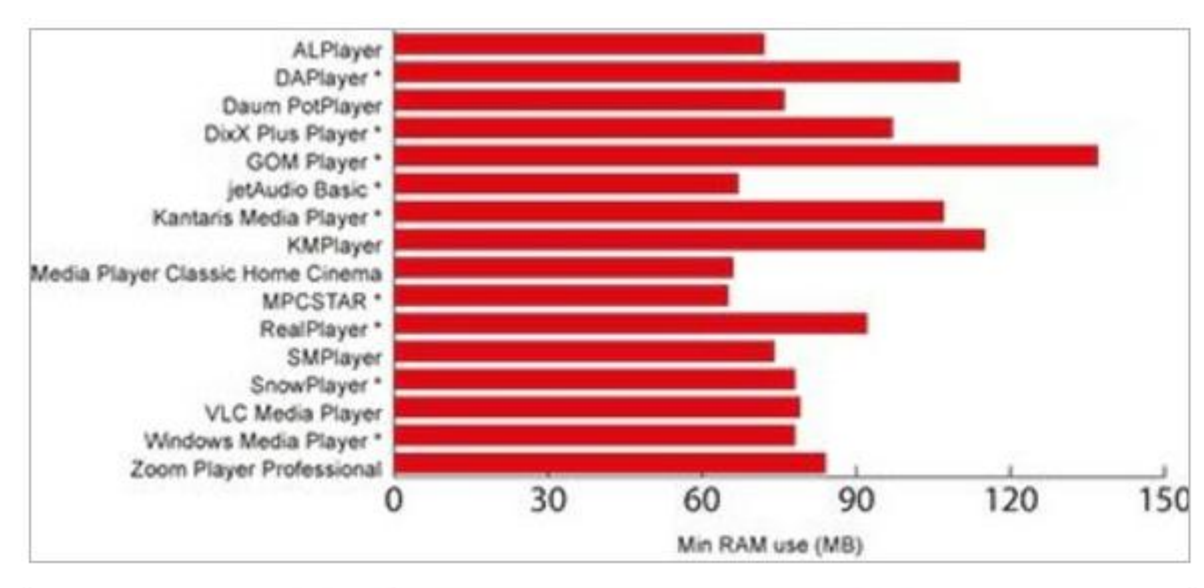
Appendix 7: VLC Media Player Performance Compared[207]

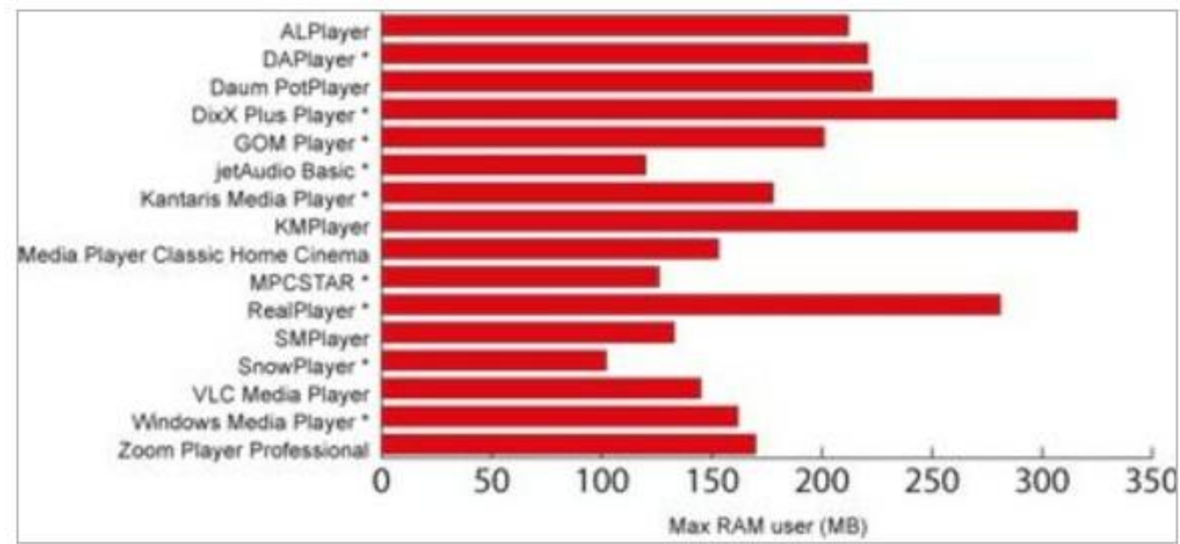






[207] Contained in: Williams, M. (2011)


ALPlayer
DAPlayer *
Daum PotPlayer
DixX Plus Player *
GOM Player *
jetAudio Basic *
Kantaris Media Player *
KMPlayer
Media Player Classic Home Cinema
MPCSTAR *
RealPlayer *
SMPlayer
SnowPlayer *
VLC Media Player
Windows Media Player *
Zoom Player Professional
0
30
60
90
120
150
Min RAM use (MB)


ALPlayer
DAPlayer *
Daum PotPlayer
DixX Plus Player *
GOM Player *
jetAudio Basic *
Kantaris Media Player *
KMPlayer
Media Player Classic Home Cinema
MPCSTAR *
RealPlayer *
SMPlayer
SnowPlayer *
VLC Media Player
Windows Media Player *
Zoom Player Professional
0
50
100
150
200
250
300
350
Max RAM user (MB)

Appendix 8: Terminology List

Commercial software	"Any software developed to be sold to consumers."[208]
Copyfree	The copyfree attribute does not include restrictions about the licenses for derivative works and can be commercialized through relicensing of derivative works.[209]
Criterion	"A standard by which something can be judged or decided. In a typical context, there is more than one criterion under consideration and thus the plural 'critieria' is more commonly encountered."[210]
Data collection	"The collection of data from surveys, or from independent or networked locations via data capture, data entry, or data logging."[211]
Group of developers	A group of developers is defined as "several people collaborating in an informal or not industrialized way"[212].
Mature	*"The word "mature" here means an environment in which predictability is high and risk (the unknown) is low."*[213]
Metrics	"Standards of measurement by which efficiency, performance, progress, or quality of a plan, process, or product can be assessed."[214]

[208] Dictionary, LLC. (2012)
[209] Cf. Wheeler, D. A. (2007)
[210] WebFinance (2012)
[211] Ibidem
[212] Origin, A. (2006), p. 12
[213] Persse, J.R. (2001), p. 1
[214] WebFinance (2012)

Modular (Software)	"Prefabricated, self-contained, standard unit that can be combined with other different but compatible modules to assemble a wide range of varied end-products such as buildings, computers, equipment, furniture, plants, shelving, software, and structures."[215]
Non-profit organizations	"A non-profit organization is a group organized for purposes other than generating profit and in which no part of the organization's income is distributed to its members, directors, or officers."[216]
Open Source Initiative	A non-profit organization attending to promote open source software.[217]
Organization of developers	An organization of developers is defined as "a group of developers managing the software life-cycle in a formalized way"[218].
Product research	"Component of market research whereby the characteristic of a good or service, that will satisfy a recognized need or want, are identified."[219]
Project area	"The project area is the system's representation of a software project. The project area defines the project deliverables, team structure, process, and schedule."[220]

[215] WebFinance (2012)
[216] Legal Information Institute (2010)
[217] Cf. Open Source Initiative (w.y.)
[218] Ibidem
[219] WebFinance (2012)
[220] IBM (2004)

Proprietary software	"Computer programs that are exclusive property of their developers or publishers, and cannot be copied or distributed without complying with their licensing agreements. Almost all commercial (shrinkwrapped) software is proprietary, but many excellent new programs (such as Apache web server, Linux operating system, and StarOffice office suite) are non-proprietary (and free)."[221]
Risk Management	"The identification, analysis, assessment, control, and avoidance, minimization, or elimination of unacceptable risks. An organization may use risk assumption, risk avoidance, risk retention, risk transfer, or any other strategy (or combination of strategies) in proper management of future events."[222]
Strong Copyleft	To ensure that OSS remains open source, the strong copyleft attribute determines the release of derivative works under the same license as of its ancestor.[223]
Trustworthiness	"Quality of being authentic and reliable."[224]
Weak Copyleft	The weak copyleft license is a compromise between the strong copyleft and the copyfree license. It allows software components to be part of or linked with proprietary software without becoming proprietary itself and is often used for software libraries.[225]
Weighting	"Statistical technique in which a data item (such as an average) is emphasized more than other data items comprising a group or summary. A number (weight) is assigned to each data item that reflects its relative importance based on the objective of the data collection."[226]

[221] WebFinance (2012)
[222] Ibidem
[223] Cf. Meeker, H. J. (2008), p. 22 et seq.
[224] WebFinance (2012)
[225] Cf. Wheeler, D. A. (2007)
[226] WebFinance (2012)

Lists of References

List of Literature

Allen, M. (2009): The Insider´s Guide to Developing Applications in JavaScript using the Palm Majo Framework, Palm webOS, Sebastopol: O´Reilly Media

Bowden, J. (2004): Writing a report, How to prepare, write and present effective reports, Oxford: How To Books Ltd.

de Vuyst, B./Fairchild, A. (2007): Legal and Economic Justification for Software Protection. in: Handbook of Research on Open Source Software: Technological, Economic and Social Perspectives, (Ed.: St. Amant, K./Still, B.), Hershey/London: IGI Global, p. 328-339

Feller, J./Fitzgerald, B. (2002): Understanding Open Source Software Development, Harlow/London: Pearson Education Limited

Golden, B. (2005): Succeeding with Open Source, w.l.: Addison-Wesley Pearson Education

Kasse, T. (2004): Practical Insight into CMMI, London: Artech House

Kavanagh, P. (2004): Open Source Software: Implementation and Management, Burlington/Oxford: Elsevier

Kenett, R.S./Baker, E. (2010): Process Improvement and CMMI for Systems and Software, London: Taylor & Francis Ltd

Kulpa, M.K./Johnson, K.A. (2003): Interpreting the CMMI: A Process Improvement Approach, London: Taylor & Francis Ltd

Meeker, H. J. (2008): The Open Source Alternative, Understanding Risks and Leveraging Opportunities, Canada/Hoboken: John Wiley & Sons

Mettler, T./Rohner, P. (2009): Situational Maturity Models as Instrumental Artifacts for Organizational Design, in: DESRIST '09 Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology, Article No. 22, Malvern: ACM

Origin, A. (2006): Method for Qualification and Selection of Open Source software (QSOS), w.l.: QSOS.org

Paul, A.G. (2006): Liability for Defective Open Source Software. In: Proceedings of the International Research Training Groups Workshop 2006, (Ed.: Happe, J./Koziolek, H./Rohr, M./Strom, C./Warns, T), trustworthy software systems series, Volume 3, Berlin: GITO-Verlag, p. 69-70

Persse, J.R. (2001): Implementing the Capability Maturity Model, San Francisco: John Wiley & Sons Inc

Popov, D. (2005): Hands on Open Source, w.l.: lulu.com

Raymond, E.S. (2001): The Cathedral and the Bazaar, Musings on Linux and Open Source by an Accidentally Revolutionary, revised edition, Sebastopol: O’Reilly Media

Rosen, L. (2005): Open Source Licensing, Software Freedom and Intellectual Property Law, New Jersey: Prentice Hall Professional Technical Reference

Rossi, M.A. (2006): Decoding the Free/Open Source Software Puzzle: A Survey of Theoretical and Empirical Contributions, in: The Economics of Open Source Software Development, (Ed.: Bitzer, J./Schröder P.J.H.), Amsterdam/Oxford: Elsevier, p. 15-55

Russo, B. et al. (2010): Agile Technologies in Open Source Development, Hereshey/New York: Information Science Reference

Stallman, R. M. (2002): Free Software, Free Society: Selected Essays of Richard M. Stallman, (Ed.: Gay, J.), Boston: Free Software Foundation

van den Berg, K. (2005): Finding Open options, An Open Source software evaluation model with a case study on Course Management Systems, August, Tilburg, p. ii-17

List of Internet and Intranet Sources

Android Developers (2012): Developers, http://developer.android.com/, retrieval: 25.06.2012

AdroidExpert (w.y.): AdroidExpert, http://www.adroidexpert.com/, retrieval: 25.06.2012

Android open source Project (w.y.): Android open source project, http://source.android.com/, retrieval: 21.06.2012

appSpotlight (2012): Looking for something? Just type…, http://www.webosroundup.com/, retrieval: 28.06.2012

AppTornado GmbH (2012): AppBrain, http://www.appbrain.com/stats/number-of-android-apps, retrieval: 21.06.2012

Arjona L. (2012): What happened to OpenBRR (Business Readiness Rating for Open Source)?, http://larjona.wordpress.com/2012/01/06/what-happened-to-openbrr-business-readiness-rating-for-open-source/, retrieval: 04.06.2012

Belcher, J.M. (2012): SEI CMMI Services, http://www.arinc.com/products/integration_services/sei_cmmi.html, retrieval: 29. 05.2012

Berry, M. (2009): Business Readiness Rating, http://opensourceschools.org.uk/business-readiness-rating.html, retrieval: 24.06.2012

Bohling, F. (2012): VLC Player Handbuch http://www.vlc.de/VLC-Handbuch.pdf retrieval: 28.06.2012

Brand Solutions (2011): Android Support Forum, http://androidsupportforum.com/, retrieval: 25.06.2012

Bray, T. (2010): Developers, Android Application Error Reports, http://android-developers.blogspot.com/2010/05/google-feedback-for-android.html, retrieval: 25.06.2012

Cabello, P. (2007): Firefox usability study, http://mozillalinks.org/2007/02/firefox-usability-study/, retrieval: 20.06.2012

Canalys (2012): Canalys, Insight. Innovation. Impact., Google's Android becomes the world's leading smart phone platform, http://www.canalys.com/newsroom/google%E2%80%99s-android-becomes-world%E2%80%99s-leading-smart-phone-platform, retrieval: 21.06.2012

Capgemini (2012): Capgemini, Introduction to Capgemini, http://www.capgemini.com/about/capgemini/vision-and-mission/, retrieval: 10.06.2012

Carr, A. (2011): Fast Company, Five Features Apple Should Steal From HP's TouchPad, WebOS, http://www.fastcompany.com/article/hp-touchpad-webos, retrieval: 28.06.2012

Chen, B. (2012): In Flop of H.P. TouchPad, an Object Lesson for the Tech Sector, http://www.nytimes.com/2012/01/02/technology/hewlett-packards-touchpad-was-built-on-flawed-software-some-say.html?_r=1&hpw#, retrieval: 28.06.2012

CHIP Xonio Online GmbH (2012): VLC Media Player, http://www.chip.de/downloads/VLC-media-player_13005928.html, retrieval: 28.06.2012

CNET Staff (2009): VLC Media Player, http://download.cnet.com/VLC-Media-Player/3000-13632_4-10267151.html#rateit, retrieval: 28.06.2012

Computer Videos (2012): VLC Qualität von Bild und Ton verbessern, http://www.computervideos.eu/videolan-vlc-media-player/222-vlc-qualitaet-von-bild-und-ton-verbessern.html, retrieval: 28.06.2012

Constantin, L. (2011): Highly Critical Vulnerabilities Identified in VLC Media Player, http://news.softpedia.com/news/Highly-Critical-Vulnerabilities-Identified-in-VLC-Media-Player-211528.shtml, retrieval: 28.06.2012

CopyfreeNews (2011): Copyfree: Unfetter your ideas, http://copyfree.org/, retrieval: 02.07.2012

Deprez,J./Alexandre, S. (w.y.): Qualoss, http://www.qualoss.org/dissemination/DEPREZ_Compare FlOSSAssessMethodo-Camera-02.pdf, retrieval: 08.06.2012

de Silva, C. (2009): TechRepublic, 10 questions to ask when selecting open source products for your enterprise, http://www.techrepublic.com/blog/10things/10-questions-to-ask-when-selecting-open-source-products-for-your-enterprise/1232, retrieval: 16.06.2012

Dictionary, LLC. (2012): Dictionary, http://dictionary.reference.com/browse/commercial+software, retrieval: 02.07.2012

Donahue, A. (w.y.): About VLC Media Player, http://www.ehow.com/about_5082282_vlc-media-player.html, retrieval: 28.06.2012

Edgewall Software (2012): Welcome to VLC Trac, https://trac.videolan.org/vlc/, retrieval: 28.06.2012

eInfochips (2011): eInfochips, The Solutions People, http://www.einfochips.com/mobility/ips-frameworks/aqa-android-quality-assurance.php, retrieval: 25.06.2012

Erenkratz, J./Taylor, R. (2003): Supporting Distributed and Decentralized Projects: Drawing Lessons from the Open Source Community. Technical report, Institute for Software Research, 2003, http://www.erenkrantz.com/Geeks/, Research/Publications/Open-Source-Process-OSIC.pdf, retrieval: 24.06.2012

FileHippo.com (2012): VLC Media Player 2.0.1, http://www.filehippo.com/de/download_vlc/history/, retrieval: 28.06.2012

Foulke, D. (2008): Android Reports, http://androidreports.com/, retrieval: 21.06.2012

Freebase, Metaweb (2012): webOS, http://www.freebase.com/view/en/palm_webos, retrieval: 24.06.2012

Free Software Foundation (2007): GNU Operating System, Why you shouldn't use the Lesser GPL for your next library, http://www.gnu.org/licenses/why-not-lgpl.en.html, retrieval: 24.06.2012

Free Software Foundation, Inc (2012): TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION, http://www.gnu.org/licenses/gpl-2.0.html#SEC3, retrieval: 28.06.2012

Google (2011): Android 3.0 User's Guide, http://static.googleusercontent.com/external_content/untrusted_dlcp/www.google.com/en//help/hc/pdfs/mobile/AndroidUsersGuide-30-100.pdf, retrieval: 25.06.2012

Haaland, K. et al (2010): Free/Libre Open Source Quality Models- a comparison between two approaches, http://www.floss.uni-jena.de/flossmedia/dokumente/Papers/Haaland+Groven+Glot+Free+Libre+Open+Source+Quality+Models+Quality-p-45.pdf, retrieval: 24.06.2012

Hewlett-Packard Company (2012a): Open webOS, http://openwebosproject.org/overview.html, retrieval: 24.06.2012

Hewlett-Packard Company (2012b): HP webOS, Enyo released as Open Source, https://developer.palm.com/content/resources/develop/overview_of_webos/overview_of_webos_user_interface.html, retrieval: 24.06.2012

Hewlett-Packard Company (2012c): Open webOS is coming, http://openwebosproject.org/, retrieval: 28.06.2012

Hewlett-Packard Company (2012d): Enyo, an object oriented, cross-platform JavaScript framework, http://enyojs.com/#documentation, retrieval: 28.06.2012

Hoepman, J./ Jacobs, B. (2005): Software Security Through Open Source. Technical report, Institute for Computing and Information Sciences, Radboud University Nijmegen, 2005. http://www.cs.ru.nl/~jhh/publications/oss-acm.pdf. retrieval: 24.06.2012

Hyde, A. (2012): What is FLOSS?, http://en.flossmanuals.net/firefox/ch004_open-source/, retrieval: 21st June 2012

IBM (2004): Project Area, http://pic.dhe.ibm.com/infocenter/rtc/v1r0m1/topic/com.ibm.team.platform.doc/topics/c_project_area.html, retrieval: 14.06.2012

Informer Technologies, Inc. (2012): VLC media player, http://vlc-media-player.software.informer.com/, retrieval: 28.06.2012

Kempf, J.-B. (2012a): The vlc-commits Archives, http://mailman.videolan.org/pipermail/vlc-commits/, retrieval: 28.06.2012

Kempf, J.-B. (2012b): VideoLAN Wiki, http://wiki.videolan.org, retrieval: 28.06.2012

Kempf, J.-B. (2012c): VideoLAN ORGANIZATION, http://www.videolan.org/, retrieval: 28.06.2012

Legal Information Institute (2010): Cornell University Law School, Non-profit organization, http://www.law.cornell.edu/wex/Non-profit_organizations, retrieval: 02.07.2012

Lehtimaki, J. (2012): DZone, What Should Android Apps Look Like?, http://java.dzone.com/articles/what-should-android-apps-look, retrieval: 25.06.2012

Leitenberger, B. (w.y.): Modelle, Vorstellungen und Wirklichkeit, http://www.bernd-leitenberger.de/modelle.shtml, retrieval: 27.06.2012

McCann, J. (2012): 10 best tablet PCs in the world today, http://www.techradar.com/news/mobile-computing/tablets/10-best-tablet-pcs-in-the-world-today-1079603, retrieval: 28.06.2012

Mobspot (2010): Choose your phone, Connect with friends, Discover cool new apps, http://www.mobspot.com/, retrieval: 28.06.2012

Mock, U. (2008): Beratung | Entwicklung | Schulung, Zusammenfassung Android, http://www.uwe-mock.de/images/stories/dhbw/zusammenfassung_android.pdf, retrieval: 25.06.2012

Mozilla Corporation (2004): Mozilla Foundation Releases the Highly Anticipated Mozilla Firefox Web 1.0 Browser, http://www.mozilla.org/en-US/press/mozilla-2004-11-09.html, retrieval: 22nd June 2012

Mozilla Corporation (2012a): History of the Mozilla Project, http://www.mozilla.org/about/history.html, retrieval: 22nd June 2012

Mozilla Corporation (2012b): Mozilla Firefox Releases, https://www.mozilla.org/en-US/firefox/releases/, retrieval: 21st June 2012

Mozilla Corporation (2012c): Firefox Project, https://www.mozilla.org/projects/firefox/, retrieval: 21st June 2012

Mozilla Corporation (2012d): Benötigen Sie Hilfe zu Firefox?, https://support.mozilla.org/de/home, retrieval: 20th June 2012

Mozilla Developer Network (2012a): Mozilla Public License, https://www.mozilla.org/MPL/, retrieval: 21st June 2012

Mozilla Developer Network (2012b): Web Entwicklergemeinschaft, https://developer.mozilla.org/de/web, retrieval: 20th June 2012

Myerson, J.M. (2007): Achieving Capability Maturity Model Integration (CMMI) maturity level 4, https://www.ibm.com/developerworks/rational/library/07/0227_myerson/, retrieval: 07.06.2012

Network Solutions, LLC (2005): BRR Business Readiness Rating, http://www.openbrr.org/, retrieval: 04.06.2012

Norsk Regnesentral (2011): Open Source, Open Collaboration and Innovation, http://publications.nr.no/Compendium-INF5780H11.pdf, retrieval: 24.06.2012

NRC FOSS (2009): National Resource Centre for FOSS, http://www.nrcfoss.au-kbc.org.in/maturity/, retrieval: 10.06.2012

OpenBRR.org (2005): Business Readiness Rating for Open Source, http://docencia.etsit.urjc.es/moodle/file.php/125/OpenBRR_Whitepaper.pdf, retrieval: 04.06.2012

open handset alliance (2007): open handset alliance, http://www.openhandsetalliance.com/press_110507.html, retrieval: 25.06.2012

open handset alliance (w.y.): open handset alliance, http://www.openhandsetalliance.com/oha_members.html, retrieval: 21.06.2012

Open Source Initiative (w.y.): Open Source Initiative, http://www.opensource.org/, retrieval: 05.06.2012

Ortega, F./Izquierdo, D./ Coca, P. (2010): Comparing Assessment Methodologies for FLOSS OpenBRR and QSOS, http://master.libresoft.es/sites/default/files/Materiales_MSWL_2010_2011/Project%20Evaluation/materiales/2.4-MSWL_Eval-20102011-Comparing_OpenBRR_QSOS.pdf, retrieval: 08.06.2012

Palm (2012): Open Source Packages, http://opensource.palm.com/packages.html, retrieval: 28.06.2012

Palm webOS Security (w.y.): Palm webOS Security Overview for Enterprise, http://www.hpwebos.com/us/assets/pdfs/business/Palm_White Paper_Security.pdf, retrieval: 28.06.2012

phpBB Group (2007): The Videolan Forums http://forum.videolan.org/ retrieval: 28.06.2012

Polkamp, C. (2011): U Test, Where the best test, http://help.utest.com/testers/crash-courses/usability/mobile-testing-android-usability-101, retrieval: 25.06.2012

Raza, C. (2012): Do open source software developers listen to their users? http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/3640/3171, retrieval: 24.06.2012

TechMediaNetwork (2012): Top Ten Review, http://internet-browser-review.toptenreviews.com/mozilla-firefox-review.html, retrieval: 21st June 2012

The Linux Information Project (2006): Copyleft Definition, http://www.linfo.org/copyleft.html, retrieval: 02.07.2012

Van Laak, Kirsten (2012): Experton Group, Open Source, Open Systems und Open Standards, http://www.experton-group.de/press/releases/pressrelease/article/open-source-open-systems-und-open-standards.html, retrieval: 24.06.2012

VLC Media Player source code Pro! (2012): VLC Media Player source code review, https://www.facebook.com/note.php?note_id=383963194975218, retrieval: 28.06.2012

Vocus (2012): PRWeb, Online Visibility from Vocus, Android Smartphone Activations Reached 331 Million in Q1'2012 Reveals New Device Tracking Database from Signals and Systems Telecom, http://www.prweb.com/releases/2012/5/prweb9514037.htm, retrieval: 21.06.2012

Wang, H./ Wang, C. (2001): Open Source Feature, Open Source Software, Adoption: A Status Report, http://www.idi.ntnu.no/grupper/su/bibliography/pdf/OpenSource/Wang,%20Wang%20-%20Open%20Source%20Software%20Adoption%20A%20Status%20Report%20-%20X.pdf, retrieval: 24.06.2012

WebFinance (2012): Business Dictionary, http://www.businessdictionary.com/, retrieval: 02.07.2012

Webinos (2012): webOS, http://webinos.org/deliverable-d026-target-platform-requirements-and-ipr/2-6-webos-futuretext-2/, retrieval: 28.06.2012

webOS Mobile Nation (2012): PSA: Newest CM9 official nightly build delivers mic functionality, http://www.webosnation.com/, retrieval: 28.06.2012

Wheeler, D. A. (2007): The Free-Libre / Open Source Software (FLOSS) License Slide, http://www.nastyprisms.com/temp/cache/www.dwheeler.com/essays/floss-license-slide.pdf, retrieval: 24.06.2012

Williams, M. (2011): The best media player for performance 2011: The best media player is..., http://www.techradar.com/news/software/applications/the-best-media-player-for-performance-2011-683569?artc_pg=2 retrieval: 28.06.2012

WoltLab GmbH (2012): Das deutsche VLC-Forum. http://www.vlc-forum.de/, retrieval: 28.06.2012

w.a. (2005a): Business Readiness Rating Assessment Template, http://docencia.etsit.urjc.es/moodle/mod/resource/view.php?id4350, retrieval: 24.06.2012

w.a. (2005b): An introduction to Mozilla Firefox, http://opensourcearticles.com/introduction_to_firefox, retrieval: 21.06.2012

w.a. (2006): Introducing QSOS, http://www.qsos.org/?page_id=7, retrieval: 07.06.2012

w.a. (2012): SEI CMMI Process Areas, http://www.tutorialspoint.com/cmmi/cmmi-process-areas.htm, retrieval: 07. 06.2012

Yam, M. (2012): Download All Versions of Mozilla Firefox 10, http://www.tomshardware.com/news/mozilla-firefox-browser-download-esr,14595.html, retrieval: 20th June 2012

Ziff Davis Enterprise Holdings Inc. (2012): OpenBRR Launches Closed Open-Source Group, http://www.eweek.com/c/a/Linux-and-Open-Source/OpenBRR-Launches-Closed-OpenSource-Group/, retrieval: 04.06.2012

Entwicklung eines Modells zur Bewertung von Open Source Produkten hinsichtlich eines produktiven Einsatzes

Seminararbeit

vorgelegt am 04. Juli 2012

Fakultät:	**Wirtschaft**
Studiengang:	**Wirtschaftsinformatik** **International Business Information Management**
Kurs:	**WWI09I**

von

Anne Golembowska

Madeline Klink

Jana Petrovic

Daniel Prescher

Inhaltsverzeichnis

Seite

Inhaltsverzeichnis II

Abkürzungsverzeichnis III

Abbildungsverzeichnis IV

1. Einleitung 1

2. Theoretische Grundlagen 3

2.1. Definitionen 3

2.1.1. Open Source Software 3

2.1.2. Reifegrad 4

2.2. Reifegradmodelle 4

2.2.1. Qualification and Selection of Open Source Software 4

2.2.2. QualiPSo OpenSource Maturity Model 7

2.2.3. Capgemini OSMM 10

2.2.4. Navica OSMM 12

2.2.5. Vergleich und Schlussfolgerung 14

3. Kriterienspezifikation 17

3.1. Kriterienbetrachtung 17

3.2. Auswahl und Definition der finalen Kriterien 18

4. Beschreibung des Tools zur Reifegradbestimmung 26

5. Fazit 32

Anhang 33

Quellenverzeichnis 53

Abkürzungsverzeichnis

CMMI	Capability Maturity Model Integration
FLOSS	Free/Libre Open Source Software
KPI	Key Performance Indicator
OMM	Open Source Maturity Modell
QSOS	Qualification and Selection of Open Source Software

Abbildungsverzeichnis

Abb. 1: QSOS mit detaillierten Checkpunkten 5

Abb. 2: QualiPSo Open Source Maturity Modell 8

Abb. 3: Prozessablauf zur Projektanalyse 27

Abb. 4: Diagramm Alter in Jahren....................38

Abb. 5: Diagramm Fehlerrate pro Monat 38

Abb. 6: Diagramm Releaseabstand in Monaten 39

Abb. 7: Diagramm Stakeholderanzahl 39

Abb. 8: Diagramm Anzahl der Forenbeiträge insgesamt 40

Abb. 9: Diagramm Anzahl der Downloads pro Monat 40

Abb. 10: Diagramm globale Popularität 41

Abb. 11: Diagramm deutsche Popularität 41

Abb. 12: Diagramm Plazierung in com! Top 100 42

Abb. 13: Diagramm Marktanteil in Prozent 42

Abb. 14: Startbildschirm – Prozessschritt 1: Wahl der Projekte 43

Abb. 15: Prozessschritt 2: Kriterien konkretisieren....................44

Abb. 16: Prozessschritt 3: Priorisierung festlegen (1/5) 45

Abb. 17: Prozessschritt 3: Priorisierung festlegen (2/5) 46

Abb. 18: Prozessschritt 3: Priorisierung festlegen (3/5) 47

Abb. 19: Prozessschritt 3: Priorisierung festlegen (4/5) 48

Abb. 20: Prozessschritt 3: Priorisierung festlegen (5/5) 49

Abb. 21: Prozessschritt 4: Datengenerierung für Projekt 50

Abb. 22: Prozessschritt 5: Reifegradermittlung....................51

Abb. 23: Prozessschritt 6: Selektion 52

1. Einleitung

Motivation

Software spielt für den reibungslosen Geschäftsablauf in Unternehmen eine wesentliche Rolle. Da heutzutage der Kostendruck für Unternehmen stetig steigt, interessieren sich Unternehmen vermehrt für den Einsatz von Open Source Software. Eine Studie von Gartner im Jahr 2008 hat gezeigt, dass bereits vor vier Jahren 85 Prozent aller Unternehmen in Deutschland Open Source einsetzten[1]. Dies rührt daher, dass Unternehmen die wesentlichen Vorteile von Open Source erkannt haben. Im Gegensatz zu kommerzieller Software ist Open Source frei zugänglich. Des Weiteren kann der Code individuell modifiziert werden[2]. Aus diesen Gründen befassen sich Unternehmen vermehrt mit dem Gedanken, bislang im Einsatz befindliche kommerzielle Software mit Open Source Software zu ersetzen.

Das Angebot an Open Source Software ist groß. Aus diesem Grund benötigen Unternehmen ein Hilfsmittel, welches die objektive Evaluation von verschiedenen Open Source Projekten unterstützt. Dies muss allerdings flexibel sein, um an die spezifischen Anforderungen, Prozesse sowie Zielvorstellungen des Unternehmens angepasst werden zu können.

Zielsetzung

Ziel dieser Arbeit ist es, ein Hilfsmittel für Unternehmen zur Bewertung von Open Source Software zu entwickeln. Dieses Hilfsmittel wird ein Tool sein, welches vordefinierte Kriterien beinhaltet, durch welche sich der Reifegrad einer Open Source Software bestimmen lässt. Hierfür soll der Anwender die recherchierten Projektdaten in das Tool einfügen, das – in Verrechnung mit einer vordefinierten Gewichtung – automatisch einen Reifegrad bestimmt. Das Tool soll Kriterien enthalten, für die bei den meisten Produktrecherchen frei zugängliche Werte zur Verfügung stehen. Anhand des Reifegrades werden die Anwendungen vergleichbar, wodurch das Tool eine Entscheidungshilfe für die Unternehmen darstellt. Es wird sichtbar, welche Anwendungen am besten zu den Anforderungen und Prozessen des jeweiligen Unternehmens passen.

[1] Vgl. http://www.heise.de/open/meldung/Gartner-Open-Source-ist-ueberall-217214.html
[2] Grad der Modifizierbarkeit abhängig von der Open Source Lizenz.

Vorgehensweise

Um den Reifegrad einer Open Source Software bestimmen zu können, werden Kriterien benötigt. In einem ersten Schritt werden bereits vorhandene Reifegradmodelle analysiert, aus denen passende Kriterien ausgewählt und durch eigene Kriterien ergänzt werden. Im zweiten Schritt wird eine Recherche unter Betrachtung von 20 Open Source Projekten durchgeführt. Ziel dieser Recherche ist es, in Erfahrung zu bringen, für welche Kriterien frei zugängliche Werte auffindbar sind. Es wird ein breites Spektrum an Open Source Projekten analysiert, um ein möglichst repräsentatives Bild zu erhalten. Auf Grundlage dieses Ergebnisses wird die initiale Kriterienliste überarbeitet, welche die Basis für das zu entwickelnde Tool sein wird.

2. Theoretische Grundlagen

2.1. Definitionen

In den zwei folgenden Abschnitten werden die für diese Arbeit wichtigsten Begriffe definiert und erläutert.

2.1.1. Open Source Software

Das Prinzip Open Source ist durch folgende Gedanken gekennzeichnet: „Die Möglichkeit Software frei abzugeben, die Offenlegung des Quellcodes und das Recht diesen zu modifizieren.“[3]

Dadurch, dass jeder auf den Quellcode zugreifen und diesen modifizieren kann, steht die Begrifflichkeit Open Source auch für die kollektive Entwicklung einer Lösung. Dies hat den Vorteil, dass Sicherheitslücken schnell erkannt und behoben werden können. Außerdem kommen bei Open Source Software häufiger offene Standards zum Einsatz und die Software kann individuell angepasst werden.

Da die Entwickler von Open Source Software generell keine Lizenzeinnahmen erhalten, besteht allerdings die Gefahr, dass nur geringe finanzielle und menschliche Ressourcen zur Weiterentwicklung der Open Source Software zur Verfügung stehen[4]. Aus diesem Grund gewinnt das Ökosystem an Bedeutung. Dieses besteht aus Stakeholder (in diesem Fall im Speziellen die Unternehmen, welche das Open Source Projekt unterstützen und somit an der Entwicklung beitragen) und den Geschäftsmodellen, welche von diesen genutzt werden. Dieses Ökosystem kompensiert die fehlenden Lizenzeinnahmen.

Der Einsatz von Software in Unternehmen ist heutzutage unerlässlich. Diese stehen vor der Entscheidung, proprietäre Software, d. h. ohne frei zugänglichen Code (engl. Closed Source Software) oder kostenlose Software mit freizugänglichem Code (engl. Open Source Software) einzusetzen. Aufgrund des stetig steigenden Kostendrucks, entscheiden sich viele Unternehmen für den Einsatz von Open Source Software, trotz verschiedener Risiken wie zum Beispiel Ungewissheit über die Langlebigkeit und zukünftige Entwicklung.

[3] Asche, M. / Bauhus, W. / Mitschke, E. / Seel, B. (2008), S. 14
[4] Vgl. http://www.tfk.de/de/leistungen/documentation/open-source-software/open-source-vorteile.html

2.1.2. Reifegrad

Bislang existiert keine allgemeingültige Definition von Reifegrad. Zum Beispiel wird in manchen Quellen die Reife in starkem Zusammenhang mit dem Alter des Projektes betrachtet, wie auf CIO.de[5]. Dort wird nach Bestimmung der Reife eines Open Source Projektes durch einen Wert auf einer Skala von eins bis zehn direkt darauf verwiesen, dass es einige Jahre dauert bis ein Projekt eine vernünftige Qualität erreicht. Die Schlussfolgerung, dass das Alter Einfluss auf die Reife eines Projektes hat ist durchaus zulässig, jedoch soll in diesem Projekt der Begriff Reifegrad weitgreifender verstanden werden.

Der Reifegrad im Rahmen dieser Arbeit gibt an, wie stabil und unternehmensorientiert die jeweilige Software entwickelt ist. Dazu werden definierte Kriterien betrachtete, welche jedes Unternehmen nach ihren individuellen Anforderungen gewichten muss, um den Reifegrad der spezifisch betrachteten Open Source Software zu identifizieren. Dieser Prozess wird für alle zur Auswahl stehenden Open Source Lösungen vollzogen, um im Anschluss den Reifegrad jeder Software zu vergleichen und diejenige Lösung auszuwählen, die im Unternehmen eingesetzt werden soll.

2.2. Reifegradmodelle

In der Praxis existieren bereits mehrere Modelle mit Hilfe derer der Reifegrad von Open Source Software bestimmt werden kann. In den nachfolgenden Abschnitten werden vier dieser Reifegradmodelle beschrieben.

2.2.1. Qualification and Selection of Open Source Software

Die Qualification and Selection of Open Source Software (QSOS) ist eine Methode um Free/Libre Open Source Software (FLOSS) auszuwerten. Die Methode wurde von Atos-Origin, einer IT Service-Firma, entwickelt und diente ursprünglich als Leitfaden für deren Support und technologische Prüfungsdienste.

Ziel der QSOS ist es, Open Source Software von dem technischen und funktionalen Standpunkt aus zu untersuchen. Mit der Methode kann entweder eine einzelne Software untersucht werden oder aber mehrere Software-Produkte miteinander verglichen werden. Die Methode unterstützt zusätzlich ein Rahmenwerk, das es ermöglicht einen effizienten Risikomanagement-Prozess zu etablieren.

[5] Vgl. http://www.cio.de/subnet/oracle-finance/2235636/index4.html

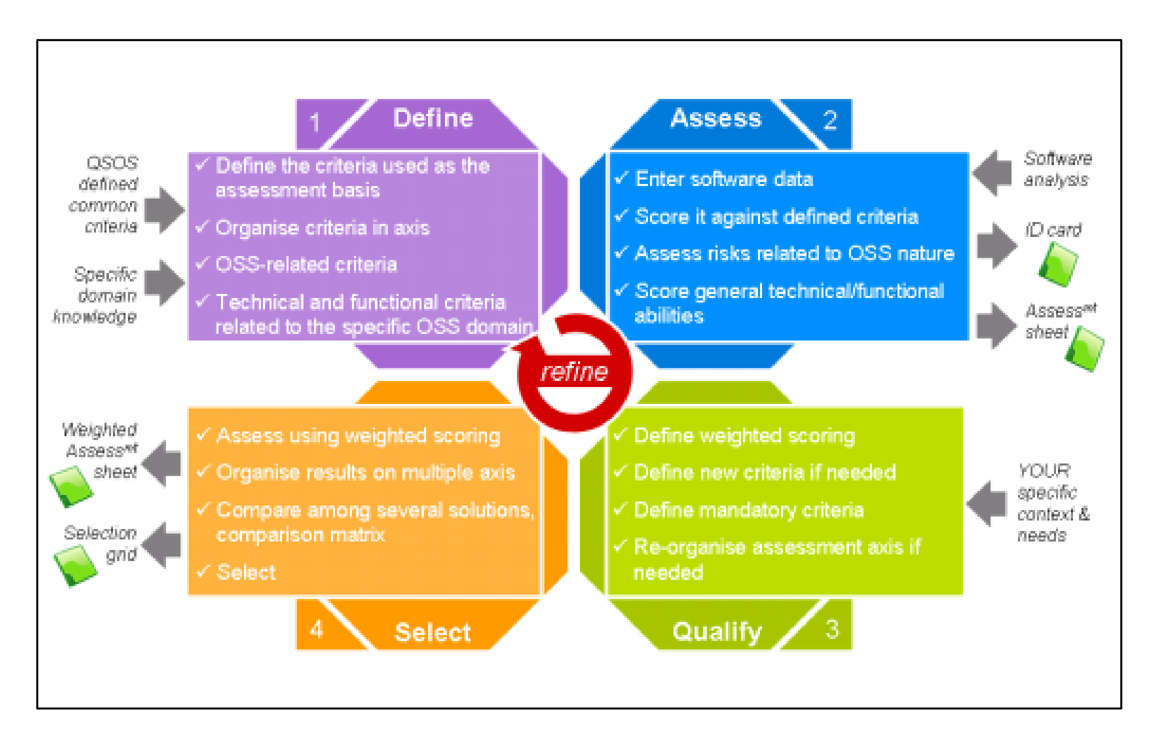


Abb. 1: QSOS mit detaillierten Checkpunkten [6]

Anhand vier voneinander unabhängiger Schritte (siehe Abb. 1) kann ein Vergleich zwischen den Produkten beziehungsweise ein einzelnes Produkt auf seinen Reifegrad hin bewertet werden. Die vier Schritte werden im Folgenden näher ausgeführt[7]:

1. Schritt: “Define”

Im ersten Schritt werden verschiedene Elemente, welche im späteren Evaluierungsprozess Verwendung finden, definiert. Bei den Elementen handelt es sich um Software-Familien, Lizenztypen, unter denen eine Open Source Software dem Nutzer übergeben wird sowie um Communities, die die Software entwickeln und unterstützen. Eine genauere Kriteriendefinition konnte zu diesem Projekt nicht gefunden werden – der Nutzer muss somit die Details selbst festlegen.

Wird das Open Source Projekt Eclipse analysiert, sind die zu definierenden Elemente beispielsweise die Größe der Community oder das Vorhandensein einer GNU-Lizenz. Dies setzt allerdings voraus, dass der Nutzer des Modells bereits ein gutes Verständnis von Open Source Software hat, um die nötigen Kriterien zu selektieren. So sollte der Nutzer sowohl technische als auch funktionale Kriterien aufstellen. Diese können sowohl Open Source spezifisch als auch Open Source unspezifische Kriterien beinhalten. Die definierten Kriterien werden als Ausgangsbasis für die Bewertung verwendet (vgl. Abbildung 1).

[6] Entnommen aus: http://en.wikipedia.org/wiki/File:QSOS-processus-en.png
[7] Vgl. Russo, B. / Sillitti, A. (2010), S. 307

2. Schritt „Assess“

Die zu untersuchenden Software Produkte werden anhand der im ersten Schritt definierten Kriterien analysiert, das heißt die Softwaredaten werden eingeben und mit den definierten Kriterien sowie den allgemeinen Open Source unspezifischen technischen und funktionalen Kriterien verglichen (vgl. Abb. 1). Diese Kriterien, anhand derer eine Bewertung des Produktes vorgenommen wird, lassen sich in drei Hauptbereiche unterteilen:

1. Abdeckung funktionaler Anforderungen
2. Risiken aus Nutzerperspektive
3. Risiken aus der Perspektive des Service Providers

Jedes Kriterium wird mit den Punkten 0 bis 2 versehen. Für den ersten Hauptbereich sieht demnach eine Bewertung folgendermaßen aus:

0: Die Funktionalität ist nicht erfüllt.
1: Die Funktionalität ist teilweise erfüllt.
2: Die Funktionalität ist voll erfüllt.

Die Unterteilung der Kriterien ist sehr generisch. Wenn für das Eclipse-Projekt beispielsweise die Communitygröße bewertet werden soll, ist es nur schwer zu sagen, wann diese Funktionalität voll oder nur teilweise erfüllt ist. Auch die Bewertung hinsichtlich der Risiken aus Nutzerperspektive oder des Service Providers ist subjektiv und ermöglicht wenig Vergleichbarkeit, wenn verschiedene Projekte gleichzeitig betrachtet werden.

3. Schritt „Qualify“

Nachdem eine Bewertung in Schritt 2 stattgefunden hat, ist der Anwender an der Reihe, den Kriterien eine Wichtigkeit beizumessen. Die Priorität ist hierbei individuell abhängig von den jeweiligen Bedürfnissen und Randbedingungen des Nutzers. Falls nötig können weitere Kriterien festgelegt werden, sodass am Ende eine Auswahl an Pflichtkriterien (engl. mandatory criteria) ausfindig gemacht werden kann (siehe Abb. 1).

Eine Gewichtung der Kriterien macht insofern Sinn, dass die Analyse an die Bedürfnisse des Nutzers angepasst werden kann. Allerdings hat der Fakt, dass die Gewichtung erst nach der Bewertung stattfindet einen signifikanten Nachteil – die ungewollte Verfälschungsgefahr. Angenommen das Eclipse-Projekt wird mit NetBeans verglichen und derjenige, der die Analyse durchführt hat eine unbewusste Tendenz hinsichtlich Eclipse. In diesem Fall weiß der Nutzer bereits wie Eclipse in den Kategorien punktet und kann unbewusst über die Gewichtung einen Vorteil von Eclipse gegenüber NetBeans erzwingen. Zudem wird dem Nutzer kein Gewichtungsfaktor vorgegeben, zum Beispiel dass er in Summe auf 100 Gewichtungs-Prozentpunkte kommen muss. Somit ist die anschließende Verrechnung unklar.

4. Schritt „Select“

Im letzten Schritt wird die Open Source Software ausgewählt, die am besten den Anforderungen des Nutzers gerecht wird (vgl. Abb. 1). Grundlage für die Entscheidung bilden die Schritte 2 und 3. Anzumerken sei an dieser Stelle, dass es sich um einen iterativen Prozess handelt. Dies bedeutet, dass die Schritte 2 und 3 beliebig oft wiederholt werden können, bevor zum nächsten Schritt übergegangen wird.

Zusammenfassend kann festgehalten werden, dass das QSOS-Modell einen großen Fokus auf den Bewertungsprozess legt und den Bewertungskriterien initial nur wenig bis gar keine Aufmerksamkeit zollt. Es werden lediglich drei Überkategorien vorgegeben. Da eines dieser Hauptkategorien die Communities sind, beinhaltet dieses Modell allerdings Open-Source-Spezifika.

Vorteile:	**Nachteile:**
+ Beschreibung des Analyseprozesses	- Ungenaue Definition der Kriterien
+ Open-Source-spezifisch	- Gewichtung erst nach der Bewertung
	- Kein Gewichtungsfaktor
	- Generische Unterteilung der Kriterien in „voll erfüllt“ etc

2.2.2. QualiPSo OpenSource Maturity Model

Das QualiPSo Open Source Maturity[8] Modell (OMM) ist ebenso wie das QSOS-Modell eine Methode um FLOSS auszuwerten. Das Modell fokussiert sich hierbei insbesondere auf den Entwicklungsprozess von FLOSS sowie dessen Verbesserung. Die Produktqualität wird hierbei in den Hintergrund gerückt.[9] Dies lässt die Vermutung zu, dass das Modell vorzugsweise für das Entwicklungsteam der Open Source Software gedacht ist und weniger für den Anwender dieser.

Das Modell ist pyramidenförmig aufgebaut und lässt sich in drei Stufen zur Bewertung des Reifegrades einer Open Source Software untergliedern (vgl. Abb. 2).

[8] dt. Reifegrad
[9] Vgl. Haaland, K. / Groven , A. / Glott, R. / Tannenberg, A. (2009), S. 8ff

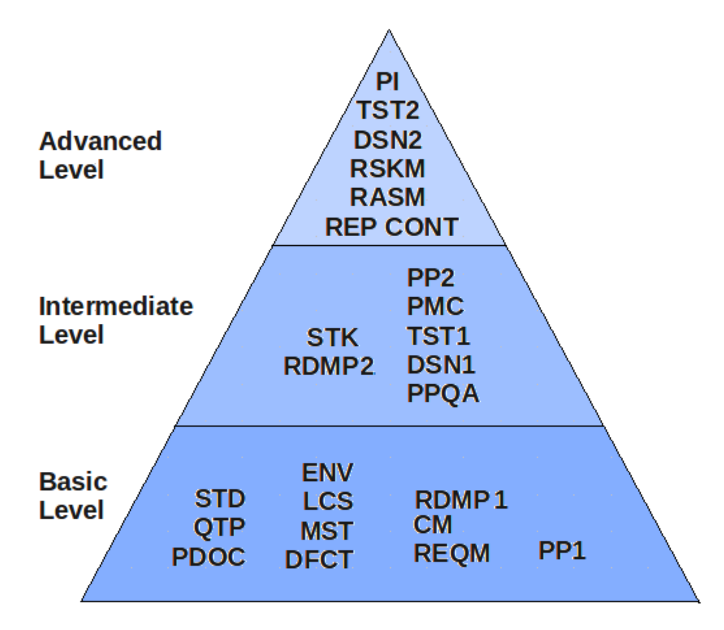


Abb. 2: QualiPSo Open Source Maturity Modell [10]

Jedes der drei Level beinhaltet verschiedene Elemente, die im Folgenden vorgestellt werden. Die Auswahl der Elemente basiert auf einer Umfrage von FLOSS-Entwicklern, -Nutzern und -Integratoren sowie der Capability Maturity Model Integration (CMMI) Prozessgebiete.[11]

Anzumerken ist, dass die Level aufeinander aufbauen und somit voneinander abhängig sind. So beinhaltet das Intermediate Level die Elemente des Basic Levels und das Advanced Level die Elemente der beiden ihm untergeordneten Level. Neben dem Beinhalten der unteren Ebenen, müssen diese auch erfüllt sein, um die Kriterien der übergeordneten Ebene anzugehen.

Basic Level:

Im untersten Level befinden sich die Basis-Kriterien für die Evaluation des Projektes. Eine vollständige Übersicht über die Abkürzungen der Kriterien ist in Anhang 1 zu finden. Wird Eclipse betrachtet, müssen für diese Ebene zum Beispiel Lizenzen (LCS) festgelegt werden und eine Projektplanung für die erste Stufe des Projektes (PP1) erstellt werden. Während Lizenzen auch für den potenziellen Nutzer von Eclipse von Wichtigkeit sind (wobei nicht nur das Vorhandensein, sondern auch die Art der Lizenzen eine Rolle spielt), ist die Projektplanung für den Einsatz des Projektes weder relevant noch recherchierbar. Daran ist ersichtlich, dass die Kriterien vorrangig für die Leitung des Eclipse-Projektes wichtig sind und nicht für den Nutzer der Software.

[10] Entnommen aus: http://en.wikipedia.org/wiki/File:OMMStructure.png
[11] Vgl. http://en.wikipedia.org/wiki/OpenSource_Maturity_Model

Intermediate Level und Advanced Level:

Auch in den nächsten beiden Ebenen des Modells setzt sich diese Tendenz – in ausgeprägter Form – fort, da die Kriterien immer allgemeiner und Projektmanagement spezifischer werden. Exemplarisch muss im Intermediate Level eingeschätzt werden, ob die Projektverfolgung und –steuerung (PMC) in ausreichendem Maße erfüllt wird oder im Advanced Level, ob das Risikomanagement durchgeführt wird. Beide Kriterien beeinflussen indirekt den Nutzer des Eclipse-Projektes, indem sie das Produktergebnis verbessern, jedoch ist es dem Nutzer unmöglich diese Kriterien aus externer Sicht einzuschätzen.

Abschließend ist festzuhalten, dass das QualiPSo-Modell zwar eine sehr genaue Vorgabe von Kriterien bereit stellt, diese allerdings eher als ein Projektmanagement-Leitfaden für die Leitung des Projektes fungieren, als eine Evaluation der Qualität und der Reife aus Sicht des potenziellen Nutzers zu ermöglichen. Weiterhin sind dies ausschließlich Kriterien, die für jedes andere Software-Projekt verwendet werden können und somit nicht Open Source-spezifisch sind. Abgesehen davon, dass das Modell für den Nutzer weniger geeignet ist, liefert es über die drei Ebenen indirekt eine Gewichtung der Kriterien mit. Der Nutzer kann somit nicht mehr selbst bestimmen, welches Kriterium ihm am wichtigsten ist. Außerdem werden keine Vorschläge hinsichtlich der Kriterien-Bewertung gemacht, zum Beispiel, ob nur zwischen „vorhanden“ und „nicht vorhanden“ differenziert wird oder ob es eine detailliertere Unterteilung gibt.

Vorteile:

+ Genaue Kriteriendefinition

Nachteile:

- Vorgegebene Gewichtung
- Kriterienbewertung nicht definiert
- Analyse für das Projektmanagement anstatt Analyse für den Nutzer hinsichtlich Reife
- Nicht Open Source-spezifisch

2.2.3.Capgemini OSMM

Das Open Source Reifegradmodell von Capgemini (OSMM = Open Source Maturity Model) wurde als Teil eines Capgemini-Prozesses entwickelt, um unabhängige Beratung für Kunden zu gewährleisten. Das Modell betrachtet hierbei zwei wichtige Aspekte von Open Source Software – zum einen das Produkt an sich und zum anderen dessen praktische Anwendung. Aus diesem Grund werden die Kriterien zur Messung des Reifegrades in Produktindikatoren und Anwendungsindikatoren unterteilt. Auf den ersten Blick macht die Unterteilung Sinn, da eine Software zwar ein gutes Produkt an sich sein kann, jedoch in der Praxis nicht anwendbar ist, da sie zum Beispiel ihren Geschäftszweck nicht erfüllt. Auf den zweiten Blick hingegen, sind Anwendungsindikatoren meist erst nach der Benutzung zu evaluieren, was dem Sinn des zu entwickelnden Reifegradmodelles entgegensteht, da dies einer Vorbetrachtung zur Auswahl des Produktes dienen soll.

Die erste der beiden Kategorien – Produktindikatoren – sind messbare Fakten, die aufzeigen, wie das Produkt dorthin gekommen ist, wo es jetzt ist und welchen Erfolg es hat Marktanteile zu gewinnen[12]. Dies ist zunächst eine sehr vage Definition, da es schwer sein dürfte, objektiv das Potenzial, Marktanteile gewinnen zu können, zu bestimmen. Dennoch hat Capgemini eine Liste an Kriterien aufgestellt, die dies ihrer Meinung nach messen. Da es eine Vielzahl an Kriterien gibt, die für den Erfolg verantwortlich sind, werden die Produktindikatoren in vier Unterkategorien aufgeteilt, die die eigentlichen Kriterien beinhalten. Diese sind[13]:

- Produkt
  - Alter
  - Lizenzen
  - Menschliche Hierarchien
  - Verkaufsargumente
  - Entwickler-Community
- Integration
  - Modularität
  - Zusammenarbeit mit anderen Produkten
  - Standards
- Verwendung
  - Support
  - Einfachheit der Einführung
- Akzeptanz
  - Nutzercommunity
  - Marktdurchdringung

[12] Vgl.http://bolsa.info.unlp.edu.ar/campamento/campamento/documentos/GB_Expert_Letter_Open_ Source _ Maturity_Model_1.5.3.pdf, S. 5
[13] Vgl. www.nrcfoss.au-kbc.org.in/maturity/

Es bleibt anzuzweifeln, ob anhand dieser Kriterien wirklich gemessen werden kann, ob ein Produkt das Potenzial, hat Marktanteile zu gewinnen. Es lässt sich nicht mit Sicherheit sagen, ob zum Beispiel eine große Nutzercommunity des Eclipse-Projektes gleichzeitig impliziert, dass es ein hohes Potenzial hat, Marktanteile zu gewinnen. Nichtsdestotrotz stellt dies auf den ersten Blick eine sehr gute und ausführliche Liste an Kriterien dar, anhand derer das Produkt hinsichtlich seiner Reife analysiert werden kann. Die Güte der Liste setzt sich unter anderem dadurch zusammen, dass für die meisten der Kriterien tatsächlich Informationen gefunden werden können. Lediglich das Kriterium „Verkaufsargumente" wurde im Vorfeld als sehr unkonkret im Hinblick auf eine Messung angesehen. Zudem beinhaltet die Kriterienliste mit den Community-Kriterien auch Eigenschaften, die Open Source-spezifisch sind.

Für die Produktindikatoren wird ein Bewertungsmaßstab zwischen 1 und 5 Punkten verwendet (1 = schlecht; 3 = durchschnittlich; 5 = exzellent), denen für jede Kategorie individuelle Vorgaben unterliegen. Die Unterteilung in fünf Stufen scheint ein gutes Maß zu sein, allerdings werden keine Angaben gemacht, wie ermittelt wird, ob ein Kriterium exzellent oder durchschnittlich ist. Am Beispiel des Eclipse-Projektes ist somit nicht klar ersichtlich, wie die Bewertung hinsichtlich der Communitygröße ist. Weiterhin wird für die Produktindikatoren keine Gewichtung bezüglich der Bedeutung für den Nutzer zugelassen. Dies hat zur Folge, dass Eclipse zum Beispiel einen hohen Reifegrad erzielt, weil es viele Community-Mitglieder hat, allerdings könnte der Fakt für den Nutzer überhaupt keine Rolle spielen.

Die zweite Kriterien-Kategorie – Anwendungskriterien – wird nicht durch Capgemini vorgegeben, sondern wird primär durch die individuellen Anforderungen des Kunden an die Open Source Software getrieben. Es gibt zwar Vorschläge für Kategorien von Capgemini, allerdings sind diese nicht verpflichtend und der Kunde gewichtet sie nach eigenen Maßstäben mit Noten zwischen 1 und 5 (1 = unwichtig; 5 = extrem wichtig). Zudem wird versucht Trends mit in die Betrachtung einzubeziehen indem aktuelle Zahlen, mit denen für die nahe (innerhalb der nächsten sechs Monate) und fernere (bis zwei Jahre) Zukunft verglichen werden. Sollte der Zukunftswert als nicht wichtig erachtet beziehungsweise kein Wert gefunden werden, wird der Wert der vorherigen Periode verwendet. Die Trendfunktion ist einerseits eine gute Idee für die Reifegradbestimmung des Produktes, andererseits wird ein Nutzer, der sich nicht perfekt im Open Source Bereich auskennt, kaum Informationen finden um die Zukunft für zum Beispiel das Eclipse-Projekt objektiv bewerten zu können.

Bevor letztendlich der finale Reifegrad des Produktes bestimmt wird, gibt der Kunde noch Kategorien an, in denen ein überdurchschnittliches Ergebnis erzielt werden muss. Danach wird der Endwert ermittelt und Capgemini gibt zudem noch einen Ratschlag, wie hoch das Ergebnis sein sollte, um die Lösung umzusetzen. Dies ist ein sehr positiver Aspekt des

Capgemeni-Modells, da ein Anwender eventuell den Endwert allein nicht richtig einschätzen kann.

Vorteile:

- + Gute Liste an Produkt-indikatoren
- + Open Source-spezifisch
- + Interpretationshilfe für Endwert

Nachteile:

- - Keine exakte Prozessbeschreibung
- - Anwendungsindikatoren eher nachträglich ermittelbar
- - Zukunftstrends schwer ermittelbar
- - Keine Gewichtung der Produkt-indikatoren

2.2.4. Navica OSMM

Das Reifegradmodell von Navica wurde ebenso wie das von Capgemini entwickelt, um Open Source Projekte zu bewerten und einzuschätzen, inwiefern sie den Ansprüchen des Kunden entsprechen. Hierfür werden drei Phasen durchlaufen bis der Reifegrad des Produktes ermittelt wird[14].

Phase 1: Bewertung wichtiger Produktelemente hinsichtlich ihrer Reife

Die zentralen Elemente, welche in dieser Phase bewertet werden, sind: Produkt/Software, Support, Dokumentation, Schulungsmaterial, Produkteinführung und Professionelle Services. Für jedes dieser Elemente werden zunächst Anforderungen, basierend auf dem Einsatzgebiet des Kunden, definiert. Für die Einführung eines J2EE Application Servers beispielsweise kann die Anforderung an Schulungsmaterial deutlich geringer sein, wenn bereits in der Vergangenheit mit ähnlichen kommerziellen Produkten gearbeitet wurde, als wenn noch kein Wissen im Umgang mit ähnlichen Produkten vorherrscht. Dies setzt allerdings voraus, dass der Nutzer schon ein Grundverständnis für Open Source Produkte hat, denn ansonsten wird es ihm schwer fallen sinnvolle Anforderungen zu definieren, da er diese eventuell aus Anforderungen an kommerzielle Software ableitet. In einem zweiten Schritt werden Ressourcen identifiziert, welche den Kunden bei der Implementierung unterstützen können, zum Beispiel Service Anbieter.

[14] Vgl. http://www.sita.co.za/FOSS/Gov_Malaysia-OSS_Guide.pdf pp. 64-66

Anschließend wird der Reifegrad für das jeweilige Element ermittelt. Die dazugehörige Skala in diesem Modell geht von „nicht existent“ bis „einsatzfertig“. Jedoch ist die Einteilung der Kriterien in diese Kategorien nicht genauer definiert. Es bedarf demnach einer subjektiven Einschätzung, wann ein bestimmtes Kriterium als „einsatzfertig“ gilt. Zudem ist die Skala sehr generisch im Hinblick dessen, dass die Anforderungen für die Kriterien erst vom Nutzer definiert werden. Falls zum Beispiel die Hauptkategorie „Produkt/Software“ mit einer Anforderung an die Community erweitert wird, macht die Einteilung „einsatzfertig“ nicht zwangsläufig Sinn, sondern vielmehr eine Bezeichnung wie „voll erfüllt“. Dies kann bei der Bewertung zu Verwirrung beim Nutzer des Modells führen. Dieser Reifegrad wird anschließend im vierten Schritt in eine Punktzahl zwischen 0 und 10 umgerechnet, welche unmissverständlich festhält, inwiefern das jeweilige Element die Anforderungen erfüllt.

Phase 2: Gewichtung der Elemente basierend auf Kundenbedürfnissen

In der zweiten Phase der Reifegradbestimmung werden die einzelnen Elemente hinsichtlich der Relevanz für den Kunden bewertet. Navica bietet hier eine Standard-Gewichtung an: Software/Produkt (4), Support (2), Dokumentation (1), Schulungsmaterial (1), Produkteinführung (1) Professionelle Services (1)[15]. Diese Standardgewichtung hat den Hintergrund, dass typischerweise der Faktor Software die höchste Gewichtung bekommt, während andere Faktoren eher unkritisch für den Erfolg sind. Allerdings wird die Option zugesprochen, dass die Gewichtung individuell angepasst werden kann. Es wird lediglich vorgeschrieben, dass die Elementgewichtung in ihrer Summe 10 ergeben muss, da der allgemeine Reifegrad auf 100 normalisiert wird. Analog zum QSOS-Modell findet auch hier die Gewichtung erst nach der Bewertung der Produkte hinsichtlich der erzielten Punktzahl statt. Dies kann ebenfalls zu ungewollter Verfälschung bei Vergleichen zwischen Produkten führen, wie es im Kapitel 2.2.1 anhand von Eclipse und NetBeans erläutert wurde. Zusätzlich führt eine beispielhafte Vorgabe einer Gewichtung wie es in diesem Modell gegeben ist zu einer Beeinflussung des Nutzers, was nicht im Sinne des Modells stehen sollte, wenn es darum geht, die Analyse auf die Bedürfnisse des Kunden anzupassen.

[15] Russo, B. / Sillitti, A. (2010), S. 306

Phase 3: Berechnung eines allgemeinen Reifegrades

Nachdem alle Elemente hinsichtlich der Bedeutung gewichtet wurden, wird nun der allgemeine Reifegrad ermittelt. Hierfür werden alle Elemente aufsummiert, so dass sich letztendlich ein Reifegrad zwischen 1 und 100 ergibt.

Vorteile:

+ Beschreibung des Analyseprozesses

Nachteile:

- Wenig Kriterienvorgaben
- Gewichtung erst nach der Bewertung
- Beispiel für Gewichtung gegeben (Beeinflussung)

2.2.5. Vergleich und Schlussfolgerung

Das Quality and Selection of Open Source (QSOS) Modell und das Reifegradmodell von Navica stellen den Prozess der Reifegradbestimmung in den Vordergrund, während das Capgemini Reifegradmodell und das QualiPSo Modell mehr Fokus auf die eigentlichen Kriterien legen, welche für die Analyse wichtig sind. Beide Ansätze haben positive und negative Aspekte, die für das Ergebnis der Arbeit wichtige Ansatzpunkte liefern, denn sowohl eine klare Vorgehensbeschreibung sowie eine Definition relevanter Kriterien wird als wichtig angesehen. Neben der Beschreibung der Vor- und Nachteile werden diese in Tabelle 1 dargestellt.

In den beiden Modellen, die das Vorgehen beschreiben, werden im ersten Schritt Kriterien definiert, beziehungsweise im Falle des Navica Modells werden anhand vorgegebener Überkategorien die speziellen Anforderungen definiert. Dies wird für diese Arbeit als kritisch angesehen, da für die Zielgruppe des zu entwickelnden Modells nicht vorausgesetzt werden kann, dass das nötige Vorwissen über Open Source vorhanden ist, welches es ermöglicht die wichtigsten Kriterien selbst zu finden. Aus diesem Grund wird es als sinnvoll erachtet, dem Kunden eine Liste mit Kriterien vorzugeben an welchen er sich orientieren kann. Hierfür wurden speziell die beiden kriterienorientierten Modelle hinsichtlich der Nützlichkeit ihrer vorgeschlagenen Kriterien betrachtet. Das QualiPSo-Modell stellt primär Eigenschaften vor, die für die Entwicklung von eigenen Open Source Projekten notwendig sind und weniger diejenigen Kriterien, welche für den Einsatz von Open Source relevant sind. Beispiele hierfür sind Eigenschaften wie ein Projektplan oder ein qualitativ hochwertiger Testplan. Daher wurden nur ausgewählte Kriterien aus diesem Modell für die erste Analyse verwendet: Dokumentation, Entwicklungsumgebung, Fehlermeldungen, Wartungsmodell und Configuration Management. Das Capgemini-OSMM bietet hinsichtlich des Einsatzes von Open Source Soft-

ware deutlich relevantere Kriterien, weshalb diese Liste bis auf das Kriterium „Verkaufsargumente“ komplett ausgewählt wurde. Dieser Punkt scheint sehr unkonkret im Hinblick auf eine objektive Messung.

Ein Punkt wird in beiden Modellen jedoch nicht erwähnt – das Ökosystem, bestehend aus Stakeholder und Geschäftsmodellen. Diese Kriterien sind für den Reifegrad eines Open Source Modells essentiell, denn nur mit ausreichend Support von größeren Stakeholdern kann ein längeres Bestehen des Projektes angenommen werden. Aus diesem Grund wurden die Kriterien Stakeholder und Geschäftsmodelle zusätzlich in die Betrachtung aufgenommen.

Beide Vorgehensmodelle für die Reifegradbestimmung beinhalten zusätzlich eine Gewichtung der Kriterien, um für den Kunden die Möglichkeit zu generieren, die Auswertung auf seine speziellen Bedürfnisse anzupassen. Auch das QualiPSo-Modell beinhaltet indirekt eine Bewertung – im Gegensatz zu den beiden Vorgangsmodellen allerdings eine vorgeschriebene. Im Hinblick auf das zu entwickelnde Modell zur Reifegradbestimmung wird dem Ansatz der Vorgehensmodelle gefolgt, da die Liste der Kriterien vorgegeben wird und der Kunde aber eine Möglichkeit braucht, diese für seinen speziellen Fall anzupassen. Allerdings werden in den existierenden Modellen zuerst die Daten der zu untersuchenden Software herausgesucht und eingetragen, bevor die Gewichtung vollzogen wird. Dies wird als nicht ideal angesehen, da hier eine unbewusste Manipulation des Ergebnisses stattfinden kann (vgl. Abschnitt 2.2.1). Dem kann vorgebeugt werden, indem die Gewichtung der Kriterien vor das Herausfinden der Software-Daten gezogen wird. Dies erhöht die Objektivität. Auch das Vorgeben einer Standard-Gewichtung, wie im Falle des Navica Reifegradmodelles, erscheint nicht logisch, da der Kunde automatisch in eine Richtung beeinflusst wird.

	QSOS	QualiPSo	Capgemini OSMM	Navica OSMM
Vorgangs-beschreibung	✓	x	x	✓
Genaue Kriterien-definition	x	✓	✓	x
Open Source spezifisch	✓	x	✓	x
Sonstige Vorteile			- gute Produktin-dikatoren - Interpretations-hilfe für Endwert	
Sonstige Nachteile	- Gewichtung erst nach Bewertung - Kein Gewich-tungsfaktor - Generische Un-terteilung der Kriterien in „voll erfüllt etc.)	- Vorgegebene Gewichtung - Kriterien-bewertung nicht definiert - Projekt-management-Analyse an-statt Qualitäts- und Reifeana-lyse	- Anwendungs-indikatoren nachträglich er-mittelbar - Zukunfts-trends schwer ermittel-bar - Keine Gewich-tung der Pro-duktindikatoren - Definition für Einteilung in „exzellent“ etc. fehlt	- Gewichtung erst nach Bewertung - Beispiel für Gewichtung - Definition für Einteilung in „einsatz-fähig“ etc. fehlt

Tabelle 1: Vergleich von vier Open Source Reifegradmodellen

3. Kriterienspezifikation

In den folgenden Abschnitten wird der Kriterienkatalog aus Kapitel 2 hinsichtlich ihrer Nützlichkeit für die Reifegradbestimmung untersucht und anschließend optimiert. Dieser neu definierte Kriterienkatalog dient anschließend als Grundlage für das zu entwickelnde Reifegrad-Bewertungsmodell.

3.1.Kriterienbetrachtung

Um die Kriterienauswahl zunächst zu validieren, wurden 20 Open Source Projekte auf die ausgewählten Kriterien untersucht. Diese Anzahl an Projekte wird als repräsentativ erachtet, da sowohl große, als auch kleine Open Source Projekte sowie ein großes Spektrum an Projekten betrachtet wurden. Dabei handelt es sich um Anwendungen aus den Bereichen Kollaboration, Betriebssysteme, Content Management System, Administration, Büroanwendungen und Multimedia und Softwareentwicklung. Die genaue Projektauswahl, sowie das Ergebnis der Kriterienrecherche sind im Anhang 2 zu finden. Damit wurde zum einen das Ziel verfolgt herauszufinden, für welches Kriterium ein Ergebnis gefunden werden kann. Der Gedanke dahinter ist, dass nur Kriterien zur Bestimmung des Reifegrads sinnvoll sind, zu denen ein Unternehmen bei seiner Recherche auch Ergebnisse finden kann. Zum anderen galt es als Ziel herauszufinden, welche Werte bei großen erfolgreichen Projekten für die Kriterien zutreffen und welche Werte weniger erfolgreiche Projekte erzielen. Dies dient als Grundlage für eine Einteilung der Kriterien hinsichtlich ihres Erfüllungsgrades (zum Beispiel erfüllt oder nicht erfüllt).

Bei der Untersuchung hat sich herausgestellt, dass neben dem fehlenden Zugang zu Informationen häufig nur qualitative, nicht aber quantitative Ergebnisse für die einzelnen Kriterien zu finden sind. Den Reifegrad anhand einer solchen großen Anzahl an qualitativen Ergebnissen festzumachen stellt sich als problematisch dar, da diese nur schwer und vor allem nur subjektiv gewichtet werden können.

Ein weiteres Problem, das sich bei der Recherche herausstellte war, dass einzelne Begrifflichkeiten oftmals unterschiedlich verstanden werden. Ein Beispiel dafür ist das Kriterium „Support“. Damit wurden zum Teil externe Provider, aber auch die verfügbare Community, direkte Ansprechpartner oder zur Verfügung stehende FAQs assoziiert.

Auf Grundlage dieser Erkenntnisse wurden im Folgenden Kriterien aussortiert, zu denen Informationen nur vereinzelt oder gar nicht zugänglich waren. Außerdem wurden Kriterien konkretisiert, um das Problem zu umgehen, mit einem Merkmal unterschiedliche Bedeutungen zu assoziieren. Des Weiteren wurden Kriterien teilweise umbenannt, um sie verständlicher zu machen. Eine wesentliche Veränderung gab es dahingehend, dass der Kriterienkatalog

zur Reifegradbestimmung um Kriterien ergänzt wurde, die ein quantitatives Ergebnis zurückliefern. Dies geschieht in Form von eindeutigen und konkret messbaren KPIs[16], welche die Bewertung der quantitativen Kriterien ermöglichen. Darüber hinaus wurden die Kriterien gruppiert, wodurch Ergebnisse zusammenhängender Kriterien recherchiert und gewichtet werden können.

3.2. Auswahl und Definition der finalen Kriterien

Im Folgenden werden die Kriterien definiert, um ein einheitliches Verständnis zu schaffen. Für die Einordnung der Kriterien wurde eine Skala von 0 bis 2 festgelegt, welcher die Ausprägung der Merkmale zugeordnet wird. Dabei steht der Skalenwert 2 für die bestmögliche Bewertung. Um festlegen zu können welche Merkmalsausprägung welchem Skalenwert zugeordnet wird (zum Beispiel die Ausprägung 11 Jahre des Merkmals Alter wird dem Skalenwert 1 zugeordnet), wurden die gleichen 20 Open Source Projekte wie zuvor, erneut untersucht. Das Ergebnis dieser Recherche ist im Anhang 4 zu finden.

Kategorie 1: Produkt

In der Oberkategorie „Produkt“ werden die Eigenschaften einer Open Source Software beleuchtet. Hierbei handelt es sich um die Basismerkmale des Produktes, zu denen Informationen meist leicht aufzufinden sind.

Produktalter: Das Alter gibt Auskunft darüber, wie lange sich eine Open Source Software bereits auf dem Markt befindet. Die Kategorie ist von Bedeutung, da ein höheres Alter eine höhere Garantie für eine längere Beständigkeit ist. Junge Produkte können durch eine Trendwelle populär werden und auch sehr schnell wieder vom Markt verschwinden.

Anhang 5 Abb. 4 zeigt die Verteilung der Merkmalsausprägung des Alters der 20 untersuchten Open Source Projekte. Es wird deutlich, dass die Mehrheit der untersuchten Projekte ein Alter von 10 bis 15 Jahren aufweisen. Aus diesem Grund wird diese Merkmalsausprägung dem Skalenwert 1 zugeteilt. Die Recherche hat ergeben, dass bekannte und große Projekte wie beispielsweise NetBeans ein höheres Alter aufweisen. Daher kann angenommen werden, dass ein hohes Produktalter ein Erfolgsindikator ist und daher dem Skalenwert 2 zugeteilt wird.

[16] KPI (engl.) Key Performance Indicator

Lizenzen: Open Source Produkte besitzen in der Regel eine freie Lizenz (= Open Source Lizenz). In Ausnahmefällen kann auch ein duales Lizenzsystem vorliegen. Dieses setzt sich aus einer Open Source und einer proprietäre Lizenz zusammen.

Die Lizenzart eines Produktes kann entscheidend sein, wenn es ein Unternehmen vorsieht eine Software nachträglich zu modifizieren, zu verbreiten oder Codeteile davon in ein eigenes, kommerzielles Produkt miteinzubeziehen.

Die Einteilung der Ausprägung der Lizenzen in den Skalenbereich wird durch eine Abfrage vom Anwender selbst bestimmt, da er entscheiden muss, welcher Lizenztyp für ihn besser oder schlechter ist (siehe dazu Kapitel 4 – 2. Prozessschritt)

Programmiersprache: Die gängigsten Programmiersprachen sind Java, C, C++. Die Programmiersprache ist besonders bei einer aktiven Mitarbeit am Projekt beziehungsweise Anpassung der Open Source Software von Interesse. In diesem Fall muss ein Unternehmen sicherstellen, dass es Fachkräfte besitzt, die für eine Weiterentwicklung mit der jeweiligen Programmiersprache vertraut sind.

Das Kriterium Programmiersprache muss zuerst vom Benutzer konkretisiert werden (siehe dazu Kapitel 4 – Prozessschritt). Dadurch ist es messbar in der Ausprägung „gegeben“ und „nicht gegeben“.

Plattform: Weiterhin ist zu untersuchen, auf welchen Plattformen die gewünschte Open Source Software lauffähig ist. Hierbei ist abzugleichen, ob diese Plattformen auch im Unternehmen verwendet werden oder bereit gestellt werden können.

Ein Ausnahmefall, bei dem der Punkt Plattform nicht zu untersuchen ist, könnte sein, wenn ein Unternehmen parallel zur Einführung der Open Source Software seine Systemlandschaft beziehungsweise die Betriebssysteme neu aufbaut. Hier wäre es denkbar, die Plattformen in Abhängigkeit von der verwendeten Software auszuwählen.

Das Kriterium Plattform muss zuerst vom Nutzer konkretisiert werden (Siehe dazu Kapitel 4 – Prozessschritt 2). Dadurch ist es messbar in der Ausprägung „unterstützt“ und „nicht unterstützt“.

Funktionalität: Mit diesem Begriff werden die Funktionen, welche die Open Source Software erfüllt, beschrieben. Sie ist mit dem zu erwartenden Einsatz abzugleichen. Bei der Funktionalität muss der Anwender einschätzen wie gut das Tool die Anforderungen an die Funktionalität abdeckt. Eine Anforderung an Eclipse könnte sein, dass mehrere Anwender parallel daran arbeiten können. Diese Funktionalität wird von Eclipse nicht erfüllt, weshalb der Benutzer beispielsweise nur 80% Erfüllungsgrad der Funktionalität auswählt.

Modularität: Die Modularität, auch Baukastenprinzip genannt, beschreibt die Erweiterbarkeit eines Produktes um sogenannte Module / Bauelemente. So kann ein Unternehmen in diesem Punkt untersuchen, ob Add-Ons vorhanden sind. Ein Vorteil ist, dass diese Erweiterungen beliebig austauschbar sind und sich die Änderung eines einzelnen Add-Ons nicht auf das gesamte Produkt auswirkt. Dieses Kriterium wird vom Anwender mit Wert „vorhanden" oder „nicht vorhanden" belegt.

Kompatibilität: Dies ist besonders von Interesse, wenn im Unternehmen bereits Software verwendet wird, welche zwingend mit der neu eingeführten Open Source Software zusammenarbeiten muss. Es ist demnach zu untersuchen, ob die Open Source Software die Anforderungen (zum Beispiel technischen Standards in der Kommunikation untereinander) der bereits vorhandenen Softwareprodukte erfüllt oder nicht, beziehungsweise ob gegebenenfalls Aktualisierungen vorzunehmen sind. Jede der vom Anwender ausgewählten Kompatibilitätsanforderung wird im Anschluss mit „gegeben" und „nicht gegeben" bewertet.

Fehlerrate: Die Fehlerrate lässt sich durch Zugriff auf Bug Tracking Tools der Projekte bestimmen. Es gilt, die Fehlerrate pro Monat für die zu untersuchende Open Source Software ausfindig zu machen. Hierfür bieten die meisten Bug Tracking Tools eine Query-Funktion mit denen Fehleranalysen durchgeführt werden können. Zum Beispiel können somit die Bugs, die in einem bestimmten Zeitraum veröffentlicht wurden, herausgefiltert werden. Die Fehlerrate signalisiert den Unternehmen, wie anfällig das Projekt für Fehler ist.

Im Anhang 5 Abb. 5 zeigt die Verteilung der Merkmalsausprägung der Fehlerrate der 20 untersuchten Open Source Projekte. Es wird deutlich, dass die Mehrheit der untersuchten Projekte 0 bis 49 Fehler für Juni 2012 aufwiesen. Aus diesem Grund wird dieser Merkmalsausprägung der höchste Skalenwert 2 zugeteilt. Eine geringe Fehlerrate indiziert, dass das Produkt weniger anfällig für Fehler ist. Das andere Extrem sind Projekte, die mehr als 1000 Fehler im Juni 2012 aufwiesen und demnach dem Skalenwert 0 zugeordnet werden, weil sie fehleranfälliger sind.

Releaseabstände: Darunter ist der Zeit zu verstehen, seitdem die letzte Version der Software herausgegeben wurde. Zu der Definition von neuen Versionen zählen auch Updates mittels Hotfix, Patch oder Service Pack.

Das Ergebnis der Recherche ist in Abb. 6 im Anhang 5 dargestellt. Es wird deutlich, dass die Abstände seit der letzten Version generell sehr gering sind. Besonders bei Eclipse erschien die letzte bereits vor weniger als einem Monat. Ein kurzer Releaseabstand ist von Vorteil, da das Produkt sicherheitstechnisch auf dem neusten Stand ist. Deshalb werden Produkte mit der Merkmalsausprägung kleiner als einen Monat dem Skalenwert 2, eine Merkmalsausprägung zwischen 1 und 5 Monaten dem Skalenwert 1 und Releaseabstände größer als 5 Mo-

nate dem Skalenwert 0 zugeteilt. Längere Releaseabstände weisen vor allem kleine Projekte wie beispielsweise K-Meleon auf.

Stakeholder: In der Kategorie soll untersucht werden, wie viele Unternehmen oder Einzelpersonen das jeweilige Open Source Projekt unterstützen. Bei einem Projekt mit einer hohen Anzahl an unterstützenden Stakeholder, beziehungsweise großen und wichtigen Unternehmen, besitzt eine höhere Wahrscheinlichkeit länger auf dem Markt zu bleiben, da diese Stakeholder ein langfristiges Interesse (primär durch ihre Geschäftsmodelle bedingt) an der Weiterentwicklung des Produktes haben.

In Abb. 7 im Anhang 5 ist das Ergebnis der Produktrecherche abgebildet. Projekte, die durch keinen Stakeholder unterstützt werden, bekommen den Skalenwert 0 zugeteilt. Die Recherche hat gezeigt, dass die meisten Projekte 1 bis 10 Stakeholder haben, weshalb hier der Mittelwert der Skala 1 gewählt wird. Besonders große Projekte wie beispielsweise Open Office werden von mehr als 10 Stakeholdern unterstützt und sind daher mit dem Skalenwert 2 belegt, da dadurch sichergestellt ist, dass andere Unternehmen finanziell hinter dem Open Source Projekt stehen.

Kategorie 2: Dokumentation

Die Oberkategorie „Dokumentation“ zielt darauf ab, den Aufbau, die Bestandteile und die Funktionsweise einer Software für den Anwender zu beschreiben. Unter Anwender fallen die Zielgruppen Endnutzer und Entwickler des Unternehmens, welches eine Open Source Software einführen möchte.

Codedokumentation: Eine Dokumentation des Codes ist von besonderem Interesse, wenn ein Unternehmen beabsichtigt, nachträgliche Änderungen an einer Software vorzunehmen. In diesem Fall ist es für die Entwickler unerlässlich, dass der in der ursprünglichen Version der Software verwendete Code dokumentiert wurde. Dieses Kriterium wird vom Anwender mit dem Wert „vorhanden“ oder „nicht vorhanden“ belegt.

Benutzerhandbuch: Das Benutzerhandbuch beinhaltet die Kerninformationen für die eigentlichen Benutzer der Software. Dies umfasst eine Anleitung zur Installation sowie zum Bedienen der Anwendung. Dieses Kriterium wird vom Benutzer mit dem Wert „vorhanden“ oder „nicht vorhanden“ belegt.

Sprache der Dokumentation: Die Sprache, in der die Dokumentation vorhanden ist, ist im Regelfall Englisch. Sollte es für ein Unternehmen wichtig sein, dass neben der englischen Version zusätzlich eine deutsche oder anderssprachige Ausführung vorhanden ist, so ist die geforderte Sprache in die Auswertung miteinzubeziehen.

Der Benutzer hat die Möglichkeit, drei Sprachen, in denen die Dokumentation verfasst sein soll anzugeben. Diese ausgewählten Sprachen werden im Anschluss vom Nutzer mit „vorhanden“ und „nicht vorhanden“ bewertet.

Qualität der Dokumentation: Die Qualität der Dokumentation gibt an, ob die Dokumentation für einen Endnutzer oder einen Entwickler, der die Software bearbeiten möchte, verständlich aufbereitet ist. Dies enthält alle Dokumentationsformen – von Codedokumentation bis hin zum Benutzerhandbuch.

Bei der Codedokumentation muss der Anwender einschätzen wie hoch die Qualität des Codes ist. Hierzu hat der Anwender zwischen den vorgegebenen Prozentwerten auszuwählen, welche angeben, wie qualitativ hochwertig die vorhandene Codedokumentation ist.

Kategorie 3: Support

In dieser Kategorie werden verschiedene Wege aufgeführt, über die der Anwender der Open Source Software Support erhalten kann. Support spielt für viele Unternehmen eine große Rolle, da dieser die Funktionsfähigkeit der eingesetzten Software sicherstellt. Welche Supportmöglichkeiten es gibt, wird im Folgenden erklärt:

Anzahl der Forumtopics: Eine Community stellt eine kostenlose Kommunikationsplattform für Experten und Nutzer dar. Für gewöhnlich hat jedes Open-Source Projekt eine offizielle Community. Diese bietet Nutzern die Möglichkeit Fragen zu stellen oder in bereits vorhandenen Beiträgen nach derselben Problemstellung zu suchen.

Die Anzahl an Forumtopics ist ein Indikator dafür, wie viele Fragen in der Community bereits diskutiert wurden. Da dieses Kriterium zur Oberkategorie Support gehört, spricht eine hohe Anzahl an Topics für eine gute Supportquelle, was von Vorteil für Unternehmen ist. Die Mehrheit der untersuchten Projekte (siehe Anhang 5 Abb. 8) weist eine Beitragszahl von 10.000-100.000 auf und wird daher dem Skalenwert 1 zugeordnet. Große Projekte, wie beispielsweise Eclipse, haben eine höhere Anzahl an Forumtopics. Da die Merkmalsausprägung größer als 100.000 ist, erhält sie den Skalenwert 2.

Direkter Ansprechpartner: Teilweise wird auf der offiziellen Open-Source-Homepage eine E-Mail Adresse oder eine Telefonnummer angezeigt, um persönlich Kontakt zu einem Experten aufnehmen zu können. Diese Option hat den Vorteil, dass einfache Hilfe bei individuellen Problemen geboten wird, allerdings keine Garantie über den Umfang der Hilfe und die Verfügbarkeit der Experten gegeben wird. Bei diesem Kriterium hat der Benutzer zwischen „vorhanden“ und „nicht vorhanden“ zu wählen.

Vorhandensein von FAQ: Frequently Asked Questions sind häufig auf der offiziellen Open Source Homepage zu finden und beantworten die Fragen der Nutzer, die sie sich am häufigsten stellen. Durch diese Option haben die Unternehmen die Möglichkeit, schnell eine Lösung für bereits bekannte und häufig auftretende Probleme zu finden. Bei diesem Kriterium hat der Anwender zwischen „vorhanden“ und „nicht vorhanden“ zu wählen.

Anzahl der Externer Provider: Vor allem große Open Source Projekte zeichnen sich dadurch aus, dass externe Provider individuellen Support anbieten. Der Vorteil der Existenz eines externen Providers ist, dass sich der Nutzer mit allen Problemen an den Dienstleister wenden kann, welcher sich um die Lösung für das Problem kümmert. Dadurch kann ein Unternehmen Geld und Zeit sparen. Der Nachteil der Inanspruchnahme des Dienstes eines externen Providers ist allerdings, dass dieser einen hohen Preis verlangen kann.

Projekte, die keinen externen Provider haben, erhalten den Skalenwert 0, da dadurch kein individueller Support auf diesem Weg vorhanden ist. Bei Verfügbarkeit eines externen Providers, wird der Skalenwert 1 zugeteilt. Der Skalenwert 2 wird dann vergeben, wenn mindestens 2 externe Provider zur Verfügung stehen. Dies ist von Vorteil, da das Unternehmen dadurch auswählen kann, welchen Support es in Anspruch nimmt.

Kategorie 4: Marktakzeptanz

Diese Kategorie zielt darauf ab, die Marktakzeptanz der Open Source Software festzustellen. Sie beinhaltet mehrere Faktoren – von Marktanteilen bis hin zur bloßen Bekanntheit der Software. Die Marktakzeptanz spielt eine wichtige Rolle für den Nutzer, denn eine etablierte Open Source Software bietet ein gewisses Maß an Sicherheit, dass die Software weiterentwickelt und Support über mindestens eine der genannten Möglichkeiten angeboten wird.

Anzahl der Downloads: Die Downloadanzahl ist ein Indikator für die Verbreitung der Software. Die Anwendung kann auf mehreren Seiten gedownloaded werden. Keine Quelle bietet jedoch eine genaue Downloadzahl von allen Bezugsmöglichkeiten. Aus diesem Grund wird auf einen der am weitesten verbreiteten Downloadanbieter chip.de zurückgegriffen, der die jeweilige Downloadzahl auf seiner Website veröffentlicht.

Bei Betrachtung dieses Kriteriums stellte sich die Frage, ob man die Downloadzahl auf eine Zeitperiode herunterbrechen soll. Es wurde sich dagegen entschieden, weil die Zeitperiode einen Trend impliziert, beispielsweise gibt die Downloadzahl des letzten Monats Auskunft über die aktuelle Marktentwicklung. In diesem Fall wird allerdings auf eine allgemeine Verbreitung wertgelegt, weshalb die historische Downloadzahl verwendet wird.

Das Rechercheergebnis ist in Anhang 5 Abb. 9 dargestellt. Das Schaubild zeigt, dass die meisten Projekte eine Downloadzahl von 10.000 bis 100.000 aufweisen. Diese Merkmalsausprägung wurde daher dem Skalenwert 1 zugeteilt. Lang bestehende, erfolgreiche Projekte wie beispielsweise Firefox haben eine sehr hohe Downloadzahl, die über 100.000 reicht und daher den Skalenwert 2 zugeteilt bekommt. Eine hohe Downloadzahl ist dementsprechend ein Indikator für ein erfolgreiches Projekt.

Globale oder deutschlandweite Popularität (Traffic Rank): Ein Indikator für die Aktualität und Popularität einer Open Source Software ist die Anzahl der Personen, die die offizielle Open Source Homepage besuchen. Alexa.com bietet eine Messgröße, welche die durchschnittlichen Besucher pro Tag und die durchschnittlichen Pageviews über einen Zeitraum von drei Monaten verrechnet und mit anderen Seiten in Verhältnis stellt. Diese Messgröße wird als Traffic Rank bezeichnet und kann abgerufen werden, indem der Link der Open Source Webseite (Startseite) auf Alexa.com eingegeben wird. Das Kriterium unterscheidet zwischen dem deutschen und globalen Traffic-Rank, da für manche Unternehmen lediglich der deutsche Markt relevant ist. Dieser KPI ist für potenzielle Nuter von Open Source Software wichtig, da sie dadurch erkennen, wie populär die entsprechende Open Source Software aktuell ist. Die Popularität spielt hinsichtlich der Beständigkeit eine wichtige Rolle.

Das Rechercheergebnis (siehe Anhang 8 Abb. 10 und Abb. 11) hat gezeigt, dass die Mehrzahl der untersuchten Produkte einen globalen Rank zwischen 5000 und 100.000 haben. Daher wurde für diesen Bereich der mittlere Skalenwert 1 festgelegt. Ein niedriger Traffic Rank indiziert eine höhere Popularität und wird daher mit dem Skalenwert 2 belegt. Ein Beispiel dafür ist Firefox, das einen niedrigen Traffic Rank hat und damit aktuell sehr populär ist. Für den deutschen Traffic-Rank verhält es sich ähnlich, nur dass der Rank insgesamt niedriger als global. Aus diesem Grund wurden die Produkte mit einem deutschen Traffic Rank von 1000 bis 25000 dem Skalenwert 1 zugeordnet.

Platzierung in com! Top 100 der Open Source Projekte: Unter den com!-Lesern wurde eine Umfrage durchgeführt, deren Ergebnis eine Liste der Top 100 Open Source Projekte des Jahres 2012[17] ist. Es gilt herauszufinden, auf welchem Platz die zu untersuchende Open Source Software eingeordnet ist. Hierbei handelt es sich um einen KPI, welcher den Unternehmen indiziert, wie die Open Source Software im Vergleich zu anderen aufgestellt ist.

Bei der Projektrecherche hat sich herausgestellt (vgl. Anhang 5 Abb. 12), dass lediglich 6,66 % der untersuchten Projekte unter den Top 10 eingeordnet sind. Diesen Projekten wur-

[17] Vgl. http://www.com-magazin.de/uploads/tx_commagdb/2012-01_Die_beste_Software_2012.pdf

de der höchste Skalenwert 2 zugeordnet, da die Plätze 1 bis 9 indizieren, dass das Produkt populär und erfolgreich ist. Ein Beispiel dafür ist Firefox, welches Platz 1 belegt und als erfolgreichster Browser gilt. Die Plätze 10 bis 100 werden mit dem Skalenwert 1 belegt und alle Projekte, die einen Ausprägungsgrad über 100 haben – die nicht unter den Top 100 zu finden sind – werden mit dem Skalenwert 0 belegt.

Marktanteile auf Basis von Fachpublikationen: Für einige Open Source Projekte existieren Fachpublikationen über den Marktanteil, den die entsprechende Software hat. Die Angaben über Marktanteile in Fachpublikationen hat höhere Aussagekraft als zum Beispiel das Kriterium Downloadzahl. Allerdings werden diese Angaben für kleinere Projekte schwer zu finden sein.

Das Ergebnis der Untersuchung der Projekte hinsichtlich deren Marktanteil (siehe Anhang 5 Abb. 13) hat gezeigt, dass wie bereits vermutet, für einige Projekte keine Informationen zum Marktanteil gefunden werden können. Ein großer Marktanteil indiziert ein erfolgreiches Produkt, was beispielsweise an Eclipse zu sehen ist, welches einen Marktanteil von 70% hat. Bei einer Merkmalausprägung zwischen 50% und 100% wurde daher der Skalenwert 2 zugeteilt. Piwik als Beispiel für ein kleines Projekt hat mit 1% einen sehr geringen Marktanteil und bekommt daher den Skalenwert 0. Projekte mit einem Marktanteil zwischen 10% und 50% erhalten den Skalenwert 1.

4. Beschreibung des Tools zur Reifegradbestimmung

In diesem Kapitel sollen die Theorie- und Analyseerkenntnisse vereint werden, um das Tool zur Reifegradbestimmung zu erstellen. Eine Anforderung an dieses ist die Unterstützung eines klar definierten Prozesses zur Reifegradbestimmung. Hierfür wurden auf Basis der Vorgehensmodelle in den Abschnitten 2.2.1 und 2.2.4 folgende Prozessschritte zur Berechnung des Reifegrades als wichtig empfunden.

1. Kriterien spezifizieren durch den Nutzer
2. Kriterien gewichten durch den Nutzer
3. Produktdaten sammeln durch den Nutzer
4. Berechnung des Reifegrades durch das Tool
5. Selektion des Produktes

Neben der Prozessunterstützung hat das zu entwickelnde Reifegradmodell folgende weitere Anforderungen:

- Vergleichbarkeit der Reifegradbestimmungen (→ Vergleichbarkeit der unterschiedlichen Alternativen)
- Klare Trennung von Objektivität (Messbarkeit) und Subjektivität (Anpassung an die Bedürfnisse des Nutzers)
- Eliminierung von unbewusster Manipulation
- Anwendbarkeit unabhängig von den Vorkenntnissen des Nutzers

In den folgenden Abschnitten wird der Aufbau und das Vorgehen des Tools beschrieben und erklärt, inwiefern die einzelnen Teile diese Anforderungen unterstützen.

Die Excelkomponente

Der Kern des Tools ist die Excelkomponente, welches aus mehreren Sheets besteht. Das Überblick-Sheet beinhaltet die Kriterien, Gewichtungen und Reifegrade aller zu analysierenden Projekte, während die verschiedenen Projekt-Sheets (benannt mit den jeweiligen Projektnamen) die Detailinformationen zu jedem Projekt liefern, insbesondere welchen Wert das Projekt in den jeweiligen Kriterien erzielt hat.

Der Aufbau der Sheets wurde von dem einer Nutzwertanalyse abgeleitet. „Die Nutzwertanalyse ist eine Planungsmethode zur systematischen Entscheidungsvorbereitung bei der Auswahl von Projektalternativen. Sie analysiert eine Menge komplexer Handlungsalternativen mit dem Zweck, die einzelnen Alternativen entsprechend den Präferenzen des Entscheidungsträgers bezüglich eines mehrdimensionalen Zielsystems zu ordnen.“[18]

Der Hintergrund davon ist zum einen, dass das Vorgehen bei einer Nutzwertberechnung den oben dargestellten Schritten sehr ähnelt. Zum anderen stimmt der in der Definition genannte Zweck – die Ordnung von Alternativen basierend auf Nutzerpräferenzen zur systematischen Entscheidungsfindung – mit den Anforderungen an das Reifegradmodell überein.

Der Prozess

Eine reine Excel-Lösung hat den entscheidenden Nachteil, dass es die Nutzeraktivitäten im Hinblick auf ihre Abfolge nicht strukturieren kann. Da dies jedoch neben der Vermeidung von Manipulation eine zentrale Anforderung an das Tool ist, wurde der Excel-Teil durch eine Dialogstruktur ergänzt. Diese soll den Nutzer durch den kompletten Ablauf der Reifegradbestimmung leiten. Der Prozessablauf, welcher unterstützt werden soll, ist in Abb. 3 aufgezeigt. Die einzelnen Schritte werden folgend ausführlich beschrieben.

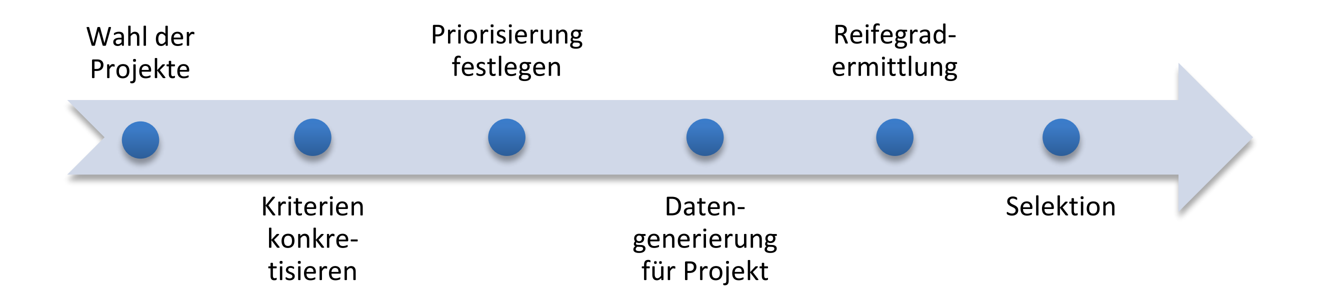


Abb. 3: Prozessablauf zur Projektanalyse

[18] http://www.laum.uni-hannover.de/ilr/lehre/Ptm/Ptm_BewNwa.htm

Schritt 1: Wahl der Projekte

Der Prozess wird durch eine Nutzerabfrage hinsichtlich der Projektauswahl initiiert (siehe Abb. 3). Dieses öffnet sich sobald das Tool gestartet wird und stellt den Anwender vor die Auswahl eine neue Analyse zu starten oder den vorherigen Vergleich beizubehalten.

Im Fall eines neuen Vergleiches muss der Anwender die Projekte, welche verglichen werden sollen, namentlich benennen (siehe Abb. 14). Hierfür hat er die Möglichkeit die Anzahl der Projekte zu verändern. Ein Limit von zehn Projekten wurde eingefügt, da es zum einen die Übersichtlichkeit stark beeinträchtigt und zum anderen soll der Nutzer sich auf einen Kern von Open Source Projekten, die idealerweise ein ähnliches Einsatzgebiet haben, konzentrieren, um eine Vergleichbarkeit zu gewährleisten. Nach Beendigung dieses Schrittes wird der Nutzer automatisch zu Schritt 2 weitergeleitet. Sollte der Anwender sich entscheiden, das bereits bestehende Projekt beizubehalten, ist dieser Prozessablauf für ihn beendet.

Schritt 2: Kriterien konkretisieren

Die Nutzwertanalyse und die vorgangsorientierten Reifegradmodelle sehen an dieser Stelle des Prozesses eine Definition der Kriterien vor. Dieses Reifegradmodell jedoch gibt die Kriterien dem Nutzer vor, da eine der Anforderungen die klare Trennung von Objektivität und Subjektivität ist. Objektivität kann nur gewährleistet werden, wenn die Kriterien eindeutig messbar sind und nicht den Bedürfnissen des Nutzers entspringen. Ein kleiner Abstrich muss hierbei allerdings gemacht werden. Einige Kriterien können in unterschiedlichen Ausprägungen betrachtet werden oder in ihrer allgemeinen Bezeichnung nicht eindeutig messbar sein. Zum Beispiel ist das Kriterium „Programmiersprache“ nicht messbar. Der Nutzer wird in diesem Prozessschritt über einen Fragebogen (siehe Abb. 15) zu einer Konkretisierung dieses und weiterer Kriterien aufgefordert. In diesem konkreten Beispiel kann er eine bestimmte Programmiersprache auswählen, in welcher das Programm geschrieben sein soll. Nach dieser Konkretisierung ist das Kriterium „Programmiersprache“ zum Beispiel in seiner Ausprägung „Java“ messbar durch eine Unterscheidung in „gegeben“ oder „nicht gegeben“.

Neben den Abstrichen in Sachen Objektivität besteht durch diesen Prozessschritt zusätzlich die Gefahr der unbewussten Manipulation durch den Nutzer, indem analog zu dem NetBeans- und Eclipse-Beispiel in Abschnitt 2.2.1. durch unbewusste Konkretisierung der Kriterien eine Manipulation des Ergebnisses hinsichtlich eines bestimmten Tools entstehen kann. Dem wurde dadurch vorgebeugt, dass dieser Schritt sehr früh in dem Prozess stattfindet, wo noch keine Recherche bezüglich der Projektdaten stattgefunden hat. Zudem wird dem Nutzer durch die Dialogstruktur nicht sofort klar, dass er durch Beantwortung der Fragen die Kriterien verändert, was die Gefahr einer Manipulation verringern soll. Nach Beendigung des Fragebogens wird automatisch zu Schritt 3 weitergeleitet.

Schritt 3: Priorisierung festlegen

In Schritt 3 hat der Anwender nun die Möglichkeit den einzelnen Kriterien eine Priorität festzulegen, um die Analyse an die spezifischen Anforderungen des Unternehmens anzupassen. Um dem Nutzer eine nicht zu lange Liste an Kriterien zu geben und ihn vor die Aufgabe zu stellen 22 Kriterien miteinander in Verhältnis zu stellen und nach Priorität zu ordnen, wurde die Kriterienliste in vier Kategorien unterteilt: Produkt, Dokumentation, Support und Marktakzeptanz. Dies hat zur Folge, dass pro Kategorie nur vier bis zehn Kriterien[19], die inhaltlich auch in Verbindung stehen, miteinander in Relation gebracht werden müssen (siehe Abb. 16 - Abb. 19). Anschließend wird eine Gewichtung der Kriterienblöcke vorgenommen, um alle Kriterien miteinander zu verrechnen (siehe Abb. 20).

Pro Kriterienblock und anschließend für die Gewichtung der Blöcke untereinander müssen 100 Prozentpunkte unter den einzelnen Punkten aufgeteilt werden. Je mehr Punkte ein Kriterium erhält, desto wichtiger ist es für den Anwender. Es wird durchaus empfohlen 0 Prozentpunkte auf Elemente zu verteilen, die dem Nutzer bei der Open Source Software egal sind. Das Tool überprüft bei der Eingabe automatisch, dass genau 100 Prozentpunkte verteilt wurden bevor es die Möglichkeit freigibt, zum nächsten Schritt fortzuschreiten.

Die Vorgabe genau 100 Punkten zu vergeben hat folgenden Sinn: Der Nutzer muss sich in einem iterativen Prozess überlegen, welche Kriterien ihm wichtiger sind als andere. Gäbe es beispielsweise keine Restriktion bezüglich der zu vergebenden Gewichtungspunkte, würde er das erste Kriterium bewerten und müsste bei der Bewertung der nächsten Kriterien nicht mehr hinterfragen, ob im Vergleich zu den anderen Kriterien das erste Kriterium eventuell zu hoch oder niedrig gewichtet wurde. Er wird somit gezwungen einen direkten Vergleich zwischen den Kriterien anzustellen und sein Ergebnis iterativ zu verfeinern.

Die im Vorfeld definierten Anforderungen an das Tool unterstützt dieser Prozessschritt in folgender Hinsicht: Zum einen wird die Subjektivität explizit gefördert, indem der Benutzer seine eigenen Präferenzen und Anforderungen abstecken kann – die subjektive Komponente ist allerdings deutlich von der objektiven Komponente in Schritt 1 getrennt. Des Weiteren birgt die Subjektivität die Gefahr der unbewussten Manipulation, wie in dem NetBeans- Eclipse-Beispiel in Abschnitt 2.2.1. beschrieben. Dem wurde entgegengewirkt, indem der Gewichtungsschritt vor den Prozess der Datenrecherche gezogen wurde und der Anwender somit noch nicht die Details über die zu analysierenden Projekte kennt um eine Manipulation

[19] Anmerkung: In Prozessschritt 2 als „egal“ definierte Kriterien werden automatisch mit 0 gewichtet und ausgegraut

vornehmen zu können. Somit sind auch keine Vorkenntnisse über Open Source an sich nötig, da es hier lediglich um Anforderungen geht. Die Vergleichbarkeit der Reifegrade von den zu analysierenden Produkten ist dadurch gegeben, dass die Gewichtung einmal festgelegt wird und für alle Produkte in gleichem Maße gilt.

Schritt 4: Datengenerierung für Projekte

In dem vierten Prozessschritt werden die benötigten Informationen zu dem Projekt gesammelt. Um die richtigen Informationen in der richtigen Form einzutragen, werden diese über eine Benutzeroberfläche abgefragt (siehe Abb. 21). Bei der Eingabemöglichkeit der Informationen zu den Kriterien wurde so oft wie möglich auf ein Drop-Down-Eingabefeld zurückgegriffen, um dem Nutzer somit deutlich zu machen, was für Informationen verlangt werden. Bei einigen Kriterien wird allerdings ein Textfeld verwendet, wo eine Zahl (ohne Tausendertrennzeichen und Dezimalpunkt) eingetragen werden muss.

Bis auf zwei Kriterien, welche vom Nutzer subjektiv bestimmt werden müssen – Qualität der Dokumentation und (Erfüllungsgrad der Anforderungen an die) Funktionalität - müssen die gesuchten Informationen vom Anwender selbst recherchiert werden. Hierfür sind folgende Quellen beispielhaft zu nennen:

- die Webseite des Herstellers / des Projektes / der Community
- Wikipedia
- Chip.de (bezüglich der Downloads)
- Com! Top 100 der Open Source Projekte
- Eine Quelle mit Informationen zum Bugtracking (zum Beispiel Trac)
- Google

Schritt 5: Reifegradermittlung

Das Exceltool übernimmt automatisch die in Schritt 4 eingefügten Daten in die Excel-Sheets und wandelt sie hinsichtlich der in Abschnitt 3.2. festgelegten Einteilung in einen Wert zwischen 0 und 2 um. Anschließend berechnet es daraus den Reifegrad (siehe Abb. 22). Hierfür wird sich an der Formel zur Nutzwertbestimmung orientiert:

1. Für jedes Kriterium wird sein erzielter Wert (zwischen 0 und 2) mit der Gewichtung multipliziert und durch 100 geteilt (da es Prozentpunkte sind).
2. Die errechneten Werte aus jedem Block werden zusammenaddiert.
3. Diese addierten Werte werden mit der Gewichtung des gesamten Blocks multipliziert und durch 100 geteilt.
4. Die Summe der Werte aller Blöcke ergibt den Reifegrad, welcher sowohl in den Sheets für die einzelnen Projekte, als auch in dem Überblick-Sheet erscheint.

Schritt 6: Selektion

Im letzten Schritt muss der Anwender ein Projekt auswählen. Dafür müssen die Reifegrade verglichen werden (siehe Abb. 23). Projekte, welche einen höheren Reifegrad als andere erzielen, sind grundsätzlich eher für einen Einsatz im Unternehmen geeignet. Allerdings muss man beachten, dass bei einem Vergleich zwischen zwei ungeeigneten Produkten eines einen besseren Wert erzielen wird und dennoch unter Umständen nicht geeignet für einen Einsatz ist. Aus diesem Grund muss auch der Wert an sich interpretiert werden.

Der höchstmöglich zu vergebene Reifegrad ist 2 und der niedrigste erzielbare Wert ist 0. Mehrere Testversuche haben ergeben, dass der Großteil der Projekte einen Reifegrad zwischen 0,7 und 1,3 erzielt. Aus diesem Grund wird zu folgender Interpretation des Reifegrades geraten:

Projekte, die ein niedrigeres Ergebnis als 1 bekommen, sind eher ungeeignet für den Einsatz im Unternehmen. Alle anderen Projekte sollten in eine engere Auswahl genommen werden, wobei lediglich die Projekte mit einem Wert von 1,2 und höher ratsame Kandidaten für den Einsatz im Unternehmen sind.

5. Fazit

Im Zuge dieses Projektes wurde ein neues Reifegradmodell entwickelt als Alternativ zu den Reifegradmodellen von Capgemini und Navica sowie QSOS und QualiPSo. Nach einer anfänglichen Betrachtung der Modelle und einer anschließenden Analyse der Kriterien wurde das finale Reifegradmodell erstellt, inklusive eines Tools, welches den Benutzer bei der Bestimmung des Reifegrades unterstützt.

Als Verbesserung gegenüber den untersuchten Modellen weist das neue Verfahren mehrere Vorteile auf. Zum einen wurde durch die geänderte Reihenfolge im Prozessablauf, die Möglichkeit zur (unbewussten) Selbstverfälschung minimiert. Gleichzeitig ermöglicht die starre Prozessorientierung durch die Dialogstruktur eine Bestimmung des Reifegrades ohne externe Hilfe. Zum anderen ist das Tool auch für im Open Source Bereich unerfahrene Nutzer anwendbar, da keine Kriterien selbst ausgewählt oder beschrieben werden müssen, da diese bereits vorgegeben werden.

Bei der Auswahl der Kriterien wurde Wert darauf gelegt, objektive Kriterien von subjektiven Einflüssen zu trennen. So wird dem Anwender eine Liste mit vordefinierten Kriterien geliefert, die sich zumeist auf messbare Merkmale bezieht. Dennoch wurde Wert darauf gelegt, dass der Anwender seine subjektiven Präferenzen in Form einer Gewichtung einbringen kann. Dieser Schritt ist jedoch deutlich von den objektiven Kriterien getrennt.

Trotz der vielen Vorteile dieses Modells im Gegensatz zu bereits vorhandenen ist es immer noch möglich das Ergebnis zu verbessern. Hierfür sollten in Zukunft die hier festgelegten Kriterien, hinsichtlich ihrer Messbarkeit noch einmal genauer betrachtet werden. Während diese Arbeit beispielsweise die Fehlerrate pro Monat untersucht, wäre es nach einer tiefergehenden Bearbeitung des Themas möglich zu analysieren, ob auch eine Fehlerrate im Verhältnisse zum Quellcode möglich wäre und wie ein solches Bewertungsschema in diese Arbeit eingepasst werden kann, um das vorhandene Kriterium zu spezifizieren und so das Ergebnis der Reifegradbestimmung zu konkretisieren.

Trotz weiterer Verbesserungsmöglichkeiten hat das entstandene Projekt die Zielvorgaben erfüllt. Das Tool ermöglicht die automatische Berechnung eines Reifegrades auf Grundlage von vorher an das System übergebende Ausprägungen. Dieser Reifegrad erlaubt die Auswahl einer kostengünstigeren Open Source Software, die den Anwender trotzdem mit den benötigten Features unterstützt.

Anhang

Anhang 1: Abkürzungen für QualiPSo 34

Anhang 2: Übersicht der Rechercheergebnisse 35

Anhang 3: Übersicht der Rechercheergebnisse 36

Anhang 4: Analyse der Kriterien 37

Anhang 5: Auswertung der Recherche 38

Anhang 6: Auswertung der Recherche 39

Anhang 7: Auswertung der Recherche 40

Anhang 8: Auswertung der Recherche 41

Anhang 9: Auswertung der Recherche 42

Anhang 10: Screenshots 43

Anhang 11: Screenshots 44

Anhang 12: Screenshots 45

Anhang 13: Screenshots 46

Anhang 14: Screenshots 47

Anhang 15: Screenshots 48

Anhang 16: Screenshots 49

Anhang 17: Screenshots 50

Anhang 18: Screenshots 51

Anhang 19: Screenshots 52

Anhang 1: Abkürzungen für QualiPSo

Abkürzung	Bedeutung
CM	Funktionierendes Konfigurationsmanagement
DFCT	Dokumentation über die Anzahl der Fehlermeldungen
DSN1	Erste Phasen für Design
ENV	Funktionierende technische Entwicklungsumgebung
LCS	Lizenzen
MST	Modell zur Wartung und Stabilität
PDOC	Dokumentation des Projektes
PI	Projektintegration
PMC	Projektverfolgung und –steuerung
PPQA	Vorhandensein einer Prozess- und Produkt-Qualitätssicherung
PP1	Planung der ersten Stufe des Projektes
PP2	Planung der zweiten Phase des Projektes
QTP	Qualitativ hochwertiger Testplan
RDMP1	Roadmap zur Verfügbarkeit und Nutzung des Produktes
RDMP2	Zweite Roadmap zur Verfügbarkeit und Nutzung des Produktes
REQM	Vollständiges Anforderungsmanagement
RSKM	Risikomanagement
STD	Anwendung von gängigen Standards
STK	Dokumentation der Beziehungen zwischen Stakeholder
TST1	Erste Testphasen
TST2	Zweite Testphase

Anhang 2: Übersicht der Rechercheergebnisse

J (J) / Nein (N)	Typo3	Zimbra	Asterisk	K-Meleon	Netbeans	Grails	Arch Linux	Pidgin	Sunbird	mySQL
Dokumentation	J	J	N	J	J	J	J	J	N	J
Standards	N	N	N	N	N	J	N		N	N
(Testplan)	N	N	N	N	N	N	N	N	N	N
Lizenzen	J	J	J	J	J	N	J	J	J	J
Entwicklungs-umgebung	N	N	N	J	N	N	N	N	N	N
Fehlermeldungen (Anzahl)	N	N	N	N	N	N	N	N	N	N
Wartungsmodell	J	N	N	N	N	N	N	N	N	N
CM	N	N	N	N	N	N	N	N	N	N
Nutzungsumfang	J	J	J	J	N	N	N	N	J	J
Stakeholder	J	J	J	J	J	J	N	J	J	J
Geschäftsmodelle	N	N	J	N	N	N	N	N	N	N
Handhabung	N	N	N	J	N	N	N	J	N	N
Integration	N	N	N	N	N	N	N	N	N	N
Produktalter	J	J	N	J	J	J	J	J	J	J
Menschliche Hierarchien	N	N	N	N	N	N	N	N	N	N
Entwickler-Community	J	N	J	J	J	J	N	N	N	N
Modularität	N	J	J	N	N	J	J	N	J	J
Kooperation mit anderen Produkten	J	J	N	J	N	N	J	J	J	J
Support	J	J	J	J	N	J	N	J	N	N
Nutzercommunity	N	N	N	N	N	N	N	N	N	N
Marktdurchdringung	J	J	N	J	N	N	J	N	J	J

Anhang 3: Übersicht der Rechercheergebnisse

	Open Office	7 Zip	Joomla	Mozilla	Apache	J Boss Application Server	Gimp	Piwik	Eclipse	Phorum
J (J) / Nein (N)										
Dokumentation	J	J	N	J	J	J	J	J	J	N
Standards	N	J	N	N	N	J	N	N	N	N
(Testplan)	N	N	N	N	N	N	N	N	N	N
Lizenzen	J	J	J	J	J	N	J	J	J	J
Entwicklungs-umgebung	J	N	N	N	N	J	N	N	J	N
Fehlermeldungen (Anzahl)	N	J	N	N	N	J	N	N	N	N
Wartungsmodell	N	N	N	N	N	N	N	N	N	N
CM	N	J	N	N	N	N	N	N	N	N
Nutzungsumfang	J	N	N	J	J	N	N	N	J	N
Stakeholder	J	J	J	J	J	J	J	J	J	J
Geschäftsmodelle	N	J	N	N	N	N	N	N	N	N
Handhabung	N	N	J	N	N	N	N	N	N	N
Integration	N	N	N	N	N	N	N	N	N	N
Produktalter	J	J	J	J	J	N	J	J	J	J
Menschliche Hierarchien	N	N	N	N	N	N	N	N	N	N
Entwickler-Community	N	J	N	J	N	J	J	N	J	N
Modularität	J	N	J	N	N	J	N	J	J	N
Kooperation mit anderen Produkten	N	N	N	J	N	N	N	N	N	N
Support	J	N	N	N	N	J	J	N	J	N
Nutzercommunity	N	N	N	N	N	J	N	J	N	J
Marktdurchdringung	J	J	J	J	J	N	J	N	J	N

Anhang 4: Analyse der Kriterien

	Open Office	7 Zip	Joomla	Firefox	Apache	J Boss Application Server	Gimp	Piwik
Produktalter (in Jahre)	10	13	7	14	17	x	17	4
Fehlerrate (pro Monat)	1.390	7	4	7.922	42	147	x	x
Releaseabstand (Datum des letzten Releases)	08.05.2012	20.06.2012	20.06.2012	15.06.2012	17.04.2012	09.05.2012	03.05.2012	05.06.2012
Stakeholder (Anzahl)	11	0	0	1	36	1	3	8
Anzahl der Forumeinträge	38.120	1.316	268.632	52.280	42.914	3.160	18.130	71.178
Anzahl der Downloads	16.505.167	9.395.212	63.838	49.792.444	93.100	x	8.979.799	1.307
Globale Popularität (gemessen am Traffic Rank der letzten drei Monate)	2.632	12.026	327	110	1.183	13.104	4.559	6.781
deutsche Popularität (gemessen am Traffic Rank der letzten drei Monate)	1.344	10.936	344	100	1.063	13.662	2.456	945
Platzierung in com! Top 100 der Open Source Projekte	3	7	>100	1	>100	>100	11	>100
Marktanteile auf Basis von Fachpublikationen (in Prozent)	22	x	30	42	63	x	x	1

	Eclipse	Phorum	Typo3	Zimbra	Asterisk	K-Meleon	Netbeans	Grails
Produktalter (in Jahre)	11	14	11	9	x	12	16	x
Fehlerrate (pro Monat)	2.043	x	63	x	0	0	x	68
Releaseabstand (Datum des letzten Releases)	27.06.2012	18.03.2011	24.05.2012	16.04.2012	27.01.2012	05.03.2010	26.04.2012	28.05.2012
Stakeholder (Anzahl)	10	0	8	1	1	1	1	1
Anzahl der Forumeinträge	227.983	17.252	43.927	49.436	x	15.300	4.769	37.580
Anzahl der Downloads	193.733	1.538	25.873	11.178	11.666	96	55	1
Globale Popularität (gemessen am Traffic Rank der letzten drei Monate)	3.903	167.245	5.158	11.019	33.888	x	6.081	36.908
deutsche Popularität (gemessen am Traffic Rank der letzten drei Monate)	3.025	90.143	430	12.768	44.730	x	5.613	51.581
Platzierung in com! Top 100 der Open Source Projekte	>100	>100	>100	>100	>100	>100	>100	>100
Marktanteile auf Basis von Fachpublikationen (in Prozent)	70	x	22	x	x	0	10	x

	Arch Linux	Pidgin	Sunbird	mySQL
Produktalter (in Jahre)	11	7	7	18
Fehlerrate (pro Monat)	88	32	x	189
Releaseabstand (Datum des letzten Releases)	x	x	30.03.2010	06.06.2012
Stakeholder (Anzahl)	x	0	1	1
Anzahl der Forumeinträge	138.785	1.382	3.153	141.387
Anzahl der Downloads	x	367	247	319
Globale Popularität (gemessen am Traffic Rank der letzten drei Monate)	13.976	35.109	298.217	164.244
deutsche Popularität (gemessen am Traffic Rank der letzten drei Monate)	9.865	25.775	22.164	8.760
Platzierung in com! Top 100 der Open Source Projekte	>100	>100	>100	>100
Marktanteile auf Basis von Fachpublikationen (in Prozent)	x	x	x	50

Anhang 5: Auswertung der Recherche

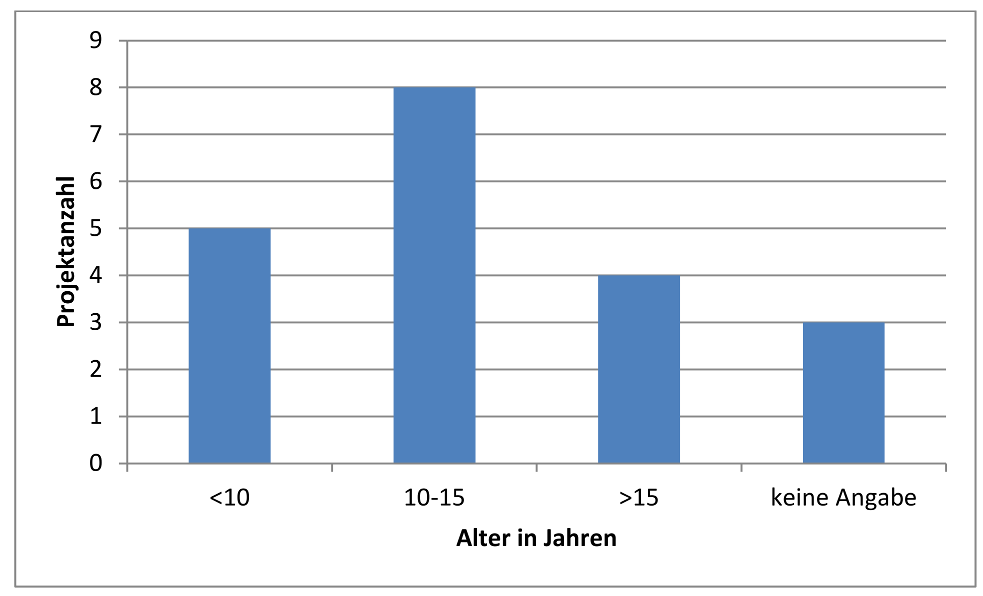


Abb. 4: Diagramm Alter in Jahren

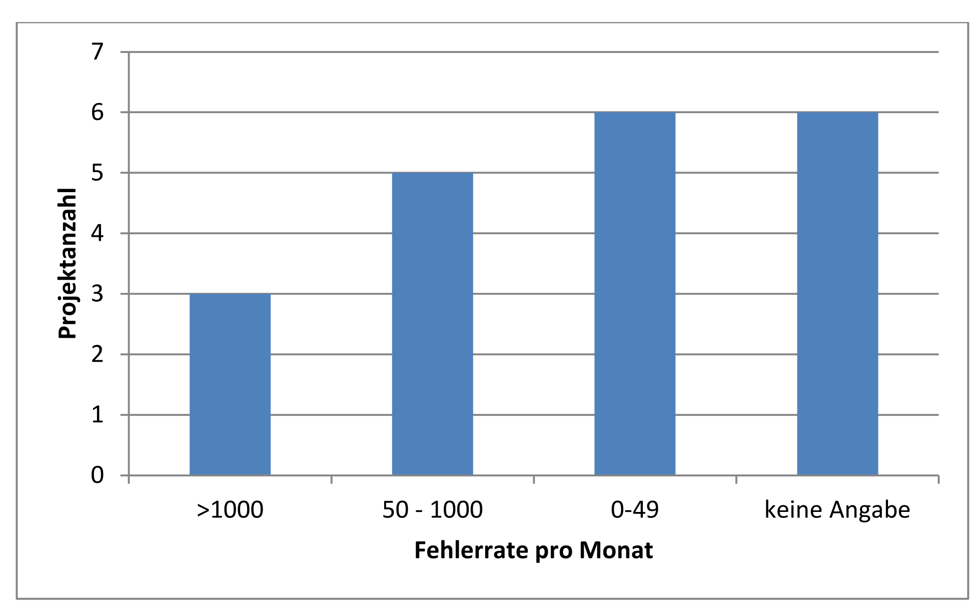


Abb. 5: Diagramm Fehlerrate pro Monat

Anhang 6: Auswertung der Recherche

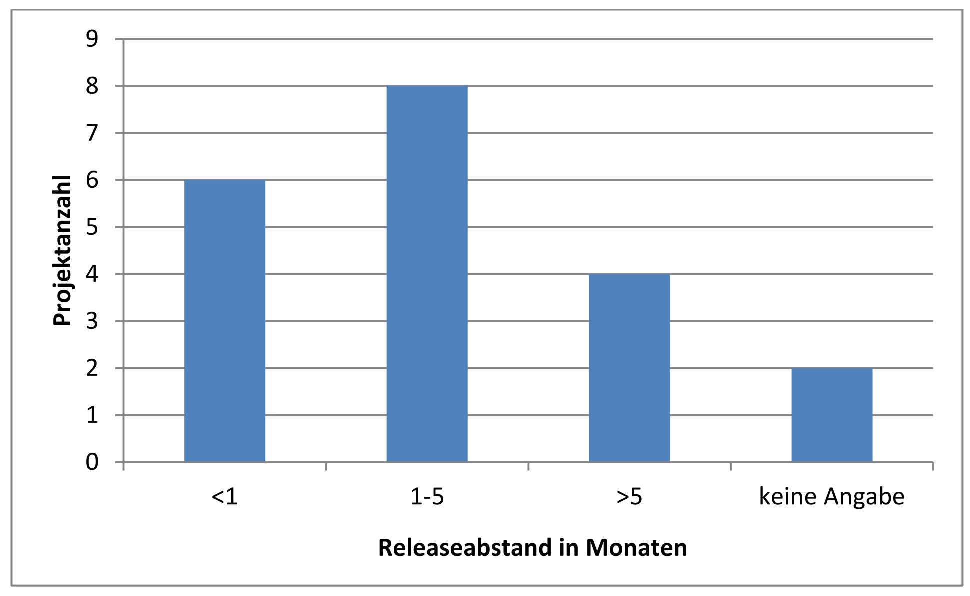


Abb. 6: Diagramm Releaseabstand in Monaten

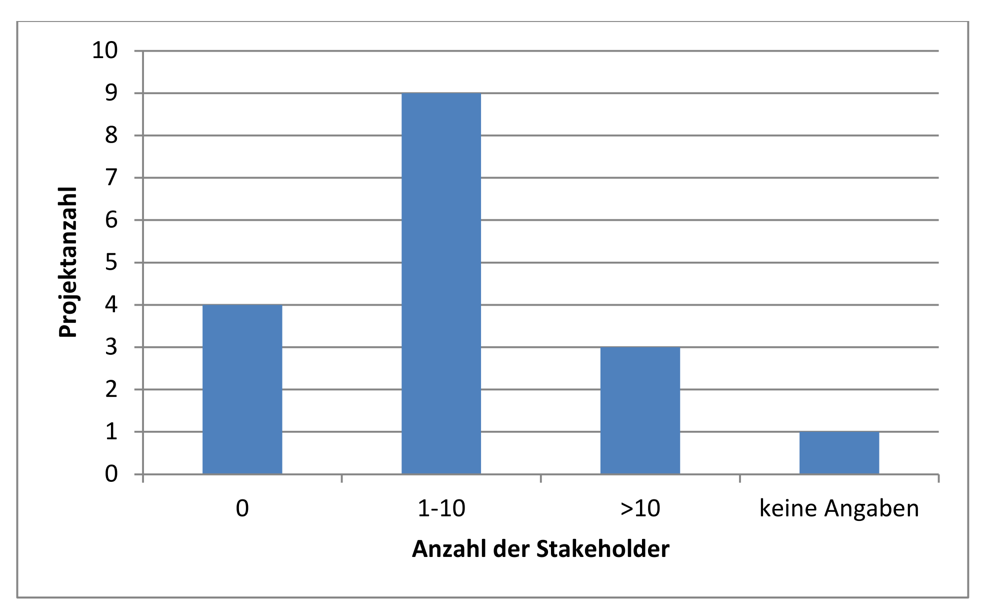


Abb. 7: Diagramm Stakeholderanzahl

Anhang 7: Auswertung der Recherche

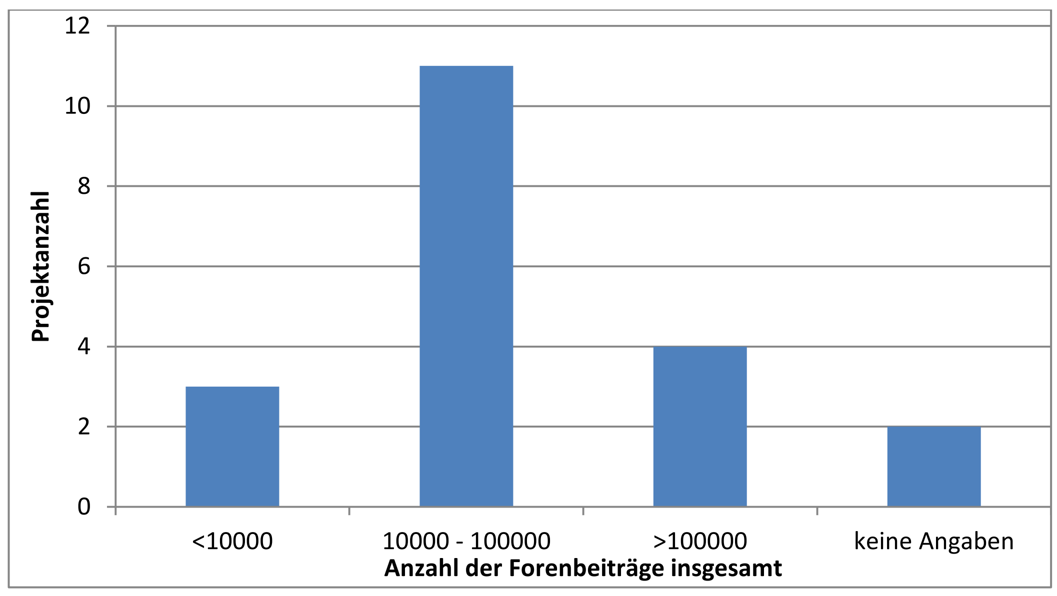


Abb. 8: Diagramm Anzahl der Forenbeiträge insgesamt

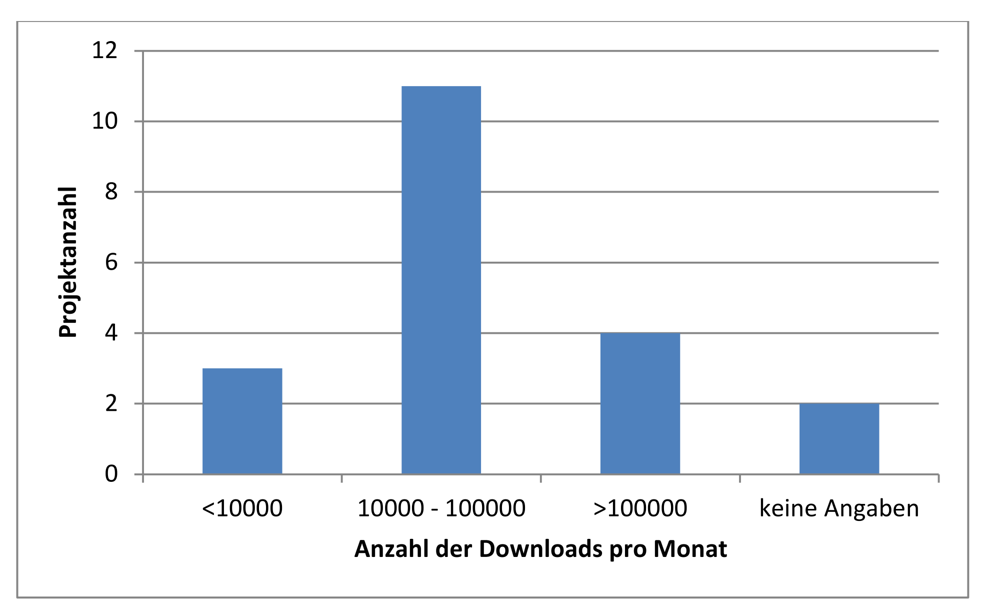


Abb. 9: Diagramm Anzahl der Downloads pro Monat

Anhang 8: Auswertung der Recherche

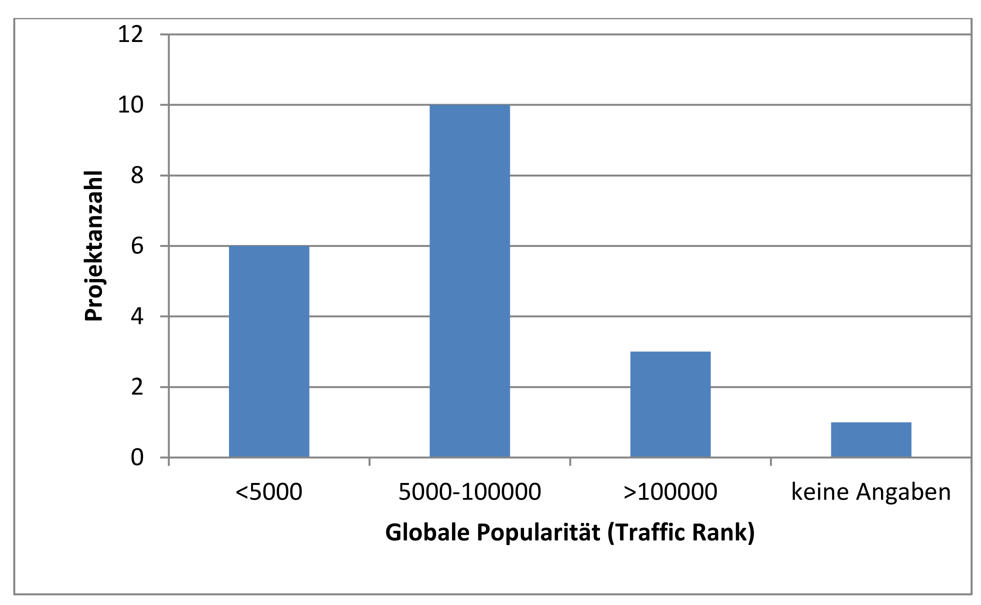


Abb. 10: Diagramm globale Popularität

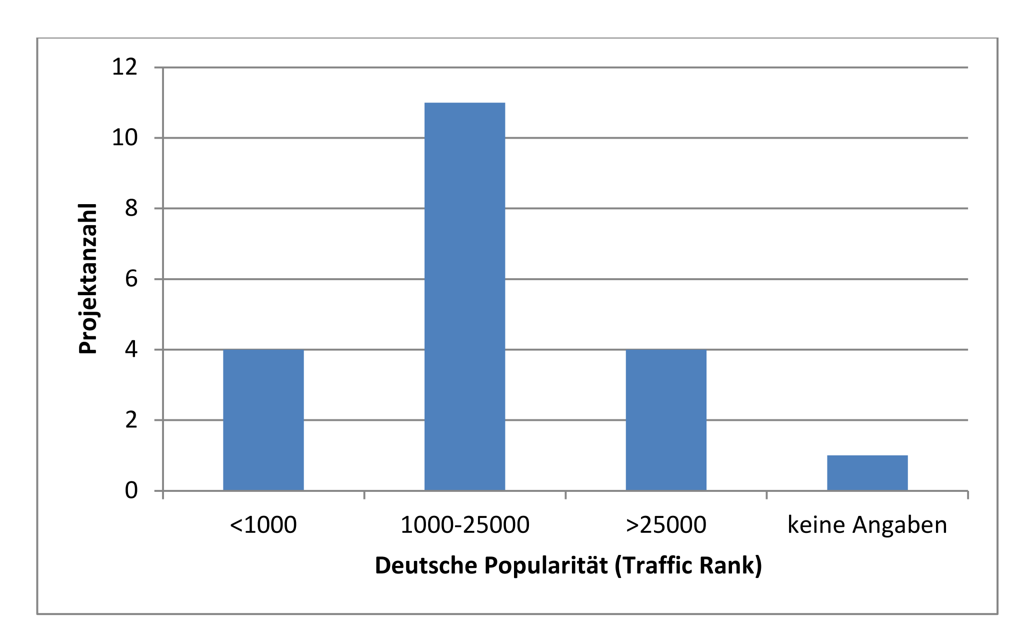


Abb. 11: Diagramm deutsche Popularität

Anhang 9: Auswertung der Recherche

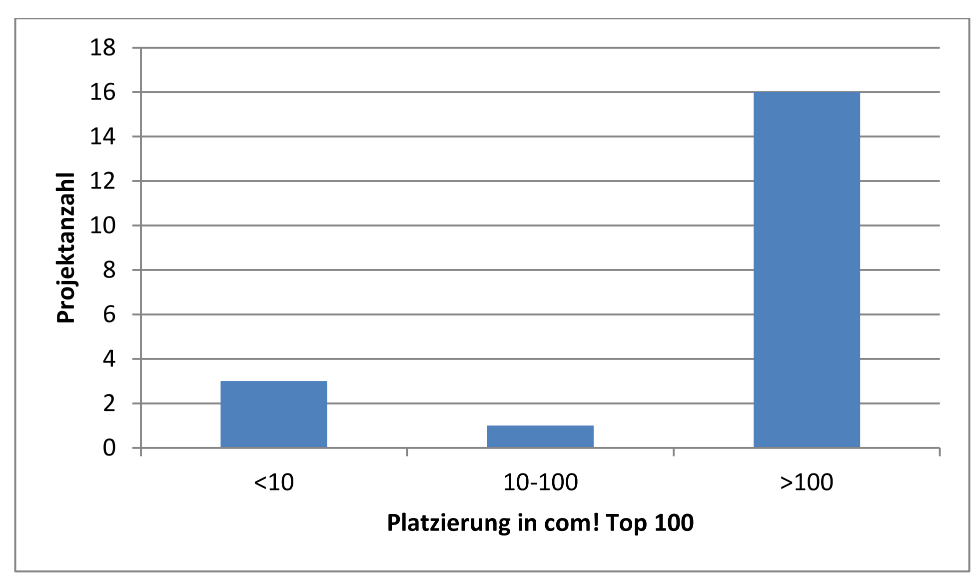


Abb. 12: Diagramm Plazierung in com! Top 100

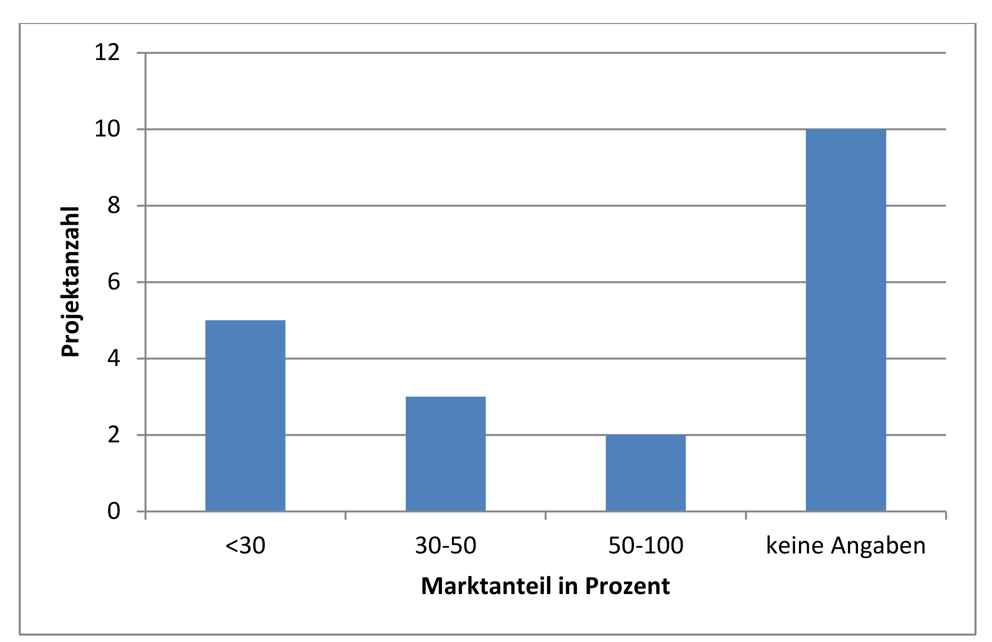


Abb. 13: Diagramm Marktanteil in Prozent

Anhang 10: Screenshots

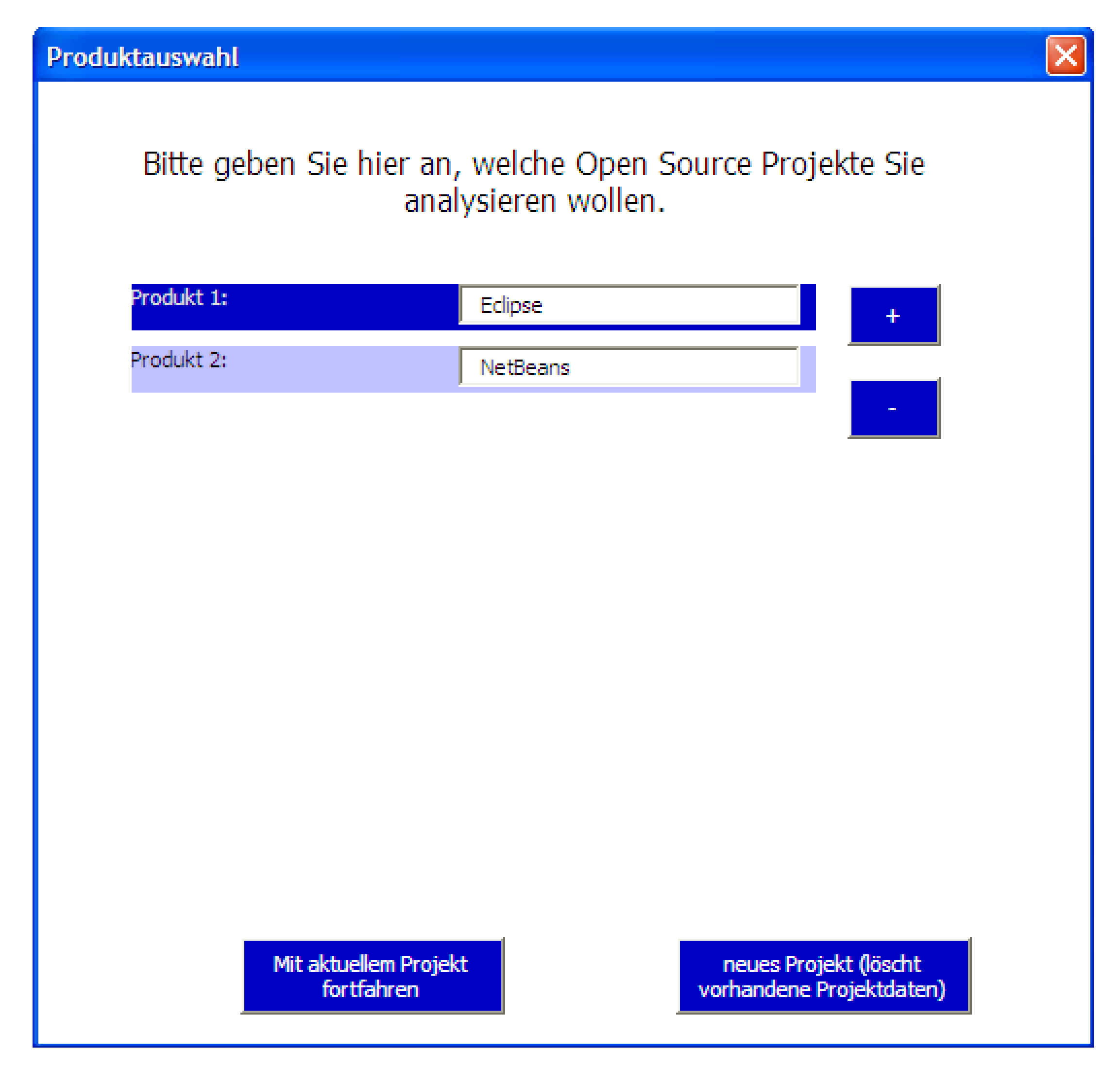


Abb. 14: Startbildschirm – Prozessschritt 1: Wahl der Projekte

Anhang 11: Screenshots

Einige Kriterien müssen konkretisiert werden - dies geschieht durch folgende Fragen:

Benötigen Sie andere Sprachen für die Dokumentation als Englisch? (•) Ja ( ) Nein

Welche Sprachen? (max. 3 - Trennung mit Semikolon) Spanisch; Deutsch

Wie wichtig sind Ihnen folgende Lizenzen? Bewertung von 0 =weniger wichtig bis 1=wichtiger (Es muss einmal die 0 und die 1 ausgewählt werden) 1 freie Lizenz 0 duale Lizenz

In welcher Programmiersprache sollte das Produkt geschrieben sein? egal Java

Auf welcher Plattform sollte das Projekt lauffähig sein? egal sonstige Windows

Mit welchen Produkten sollte das OS-Proj. kompatibel sein? (max. 3 - Trennung mit Semikolon)

Interessiert Sie das Produkt mehr hinsichtlich der Popularität auf deutschem Markt oder global gesehen? ( ) deutscher Markt (•) globaler Markt

weiter

Abb. 15: Prozessschritt 2: Kriterien konkretisieren

Anhang 12: Screenshots



Abb. 16: Prozessschritt 3: Priorisierung festlegen (1/5)

Anhang 13: Screenshots

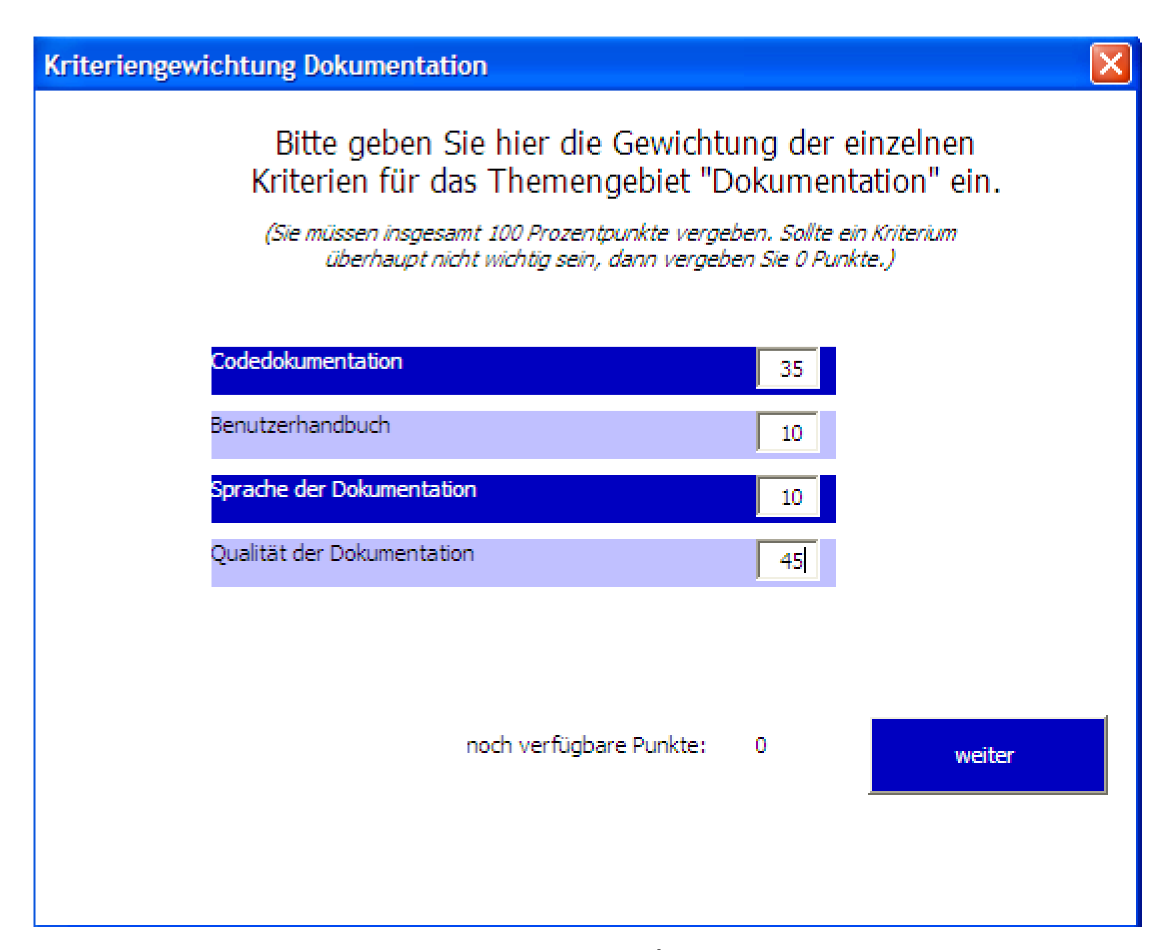


Abb. 17: Prozessschritt 3: Priorisierung festlegen (2/5)

Anhang 14: Screenshots

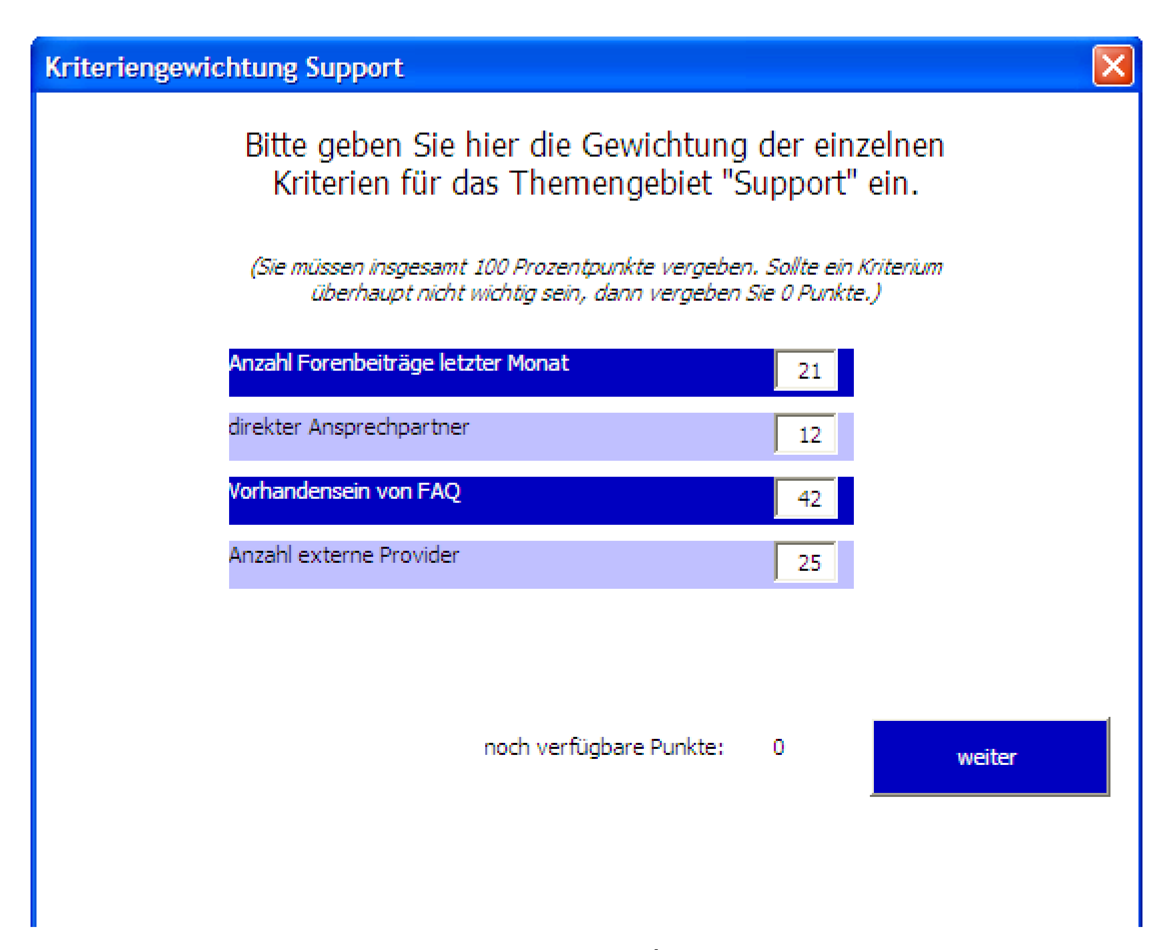


Abb. 18: Prozessschritt 3: Priorisierung festlegen (3/5)

Anhang 15: Screenshots

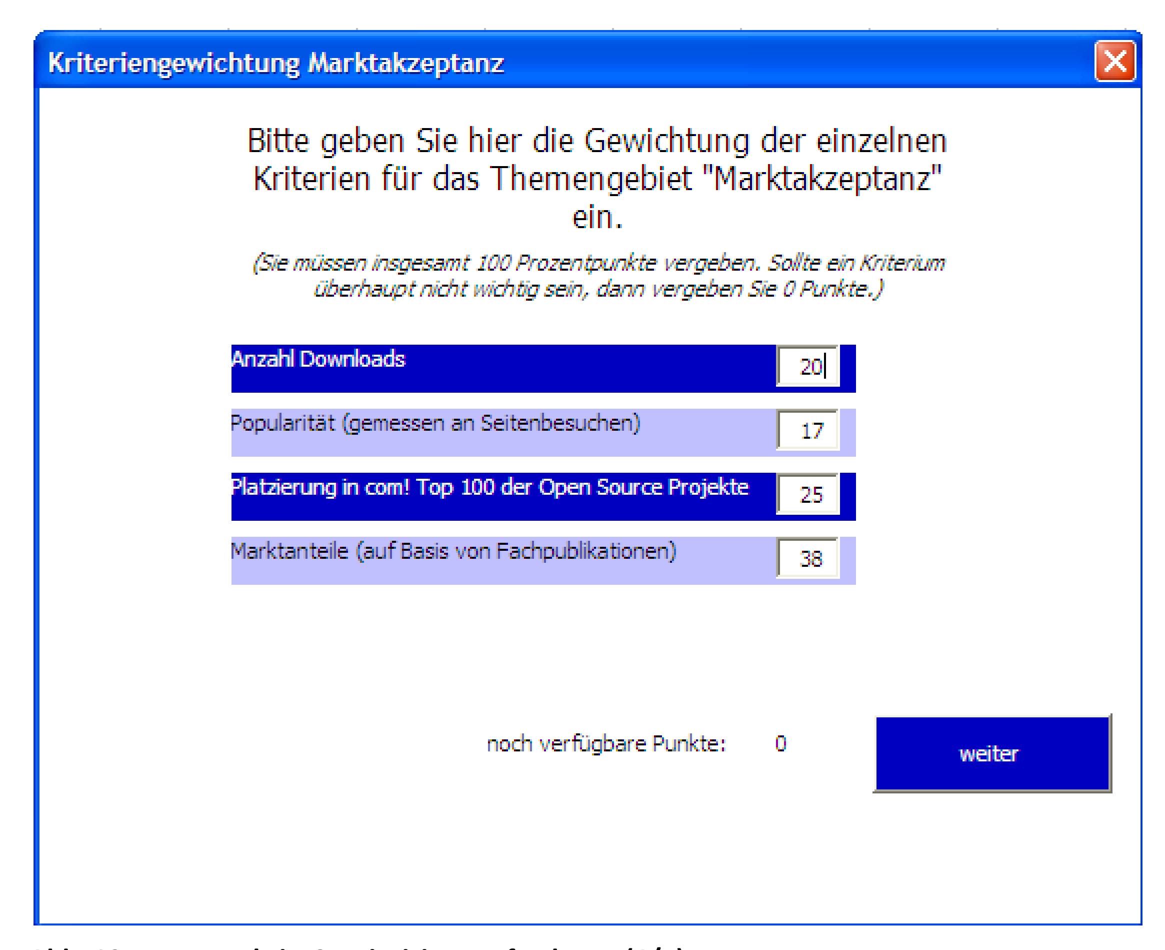


Abb. 19: Prozessschritt 3: Priorisierung festlegen (4/5)

Anhang 16: Screenshots

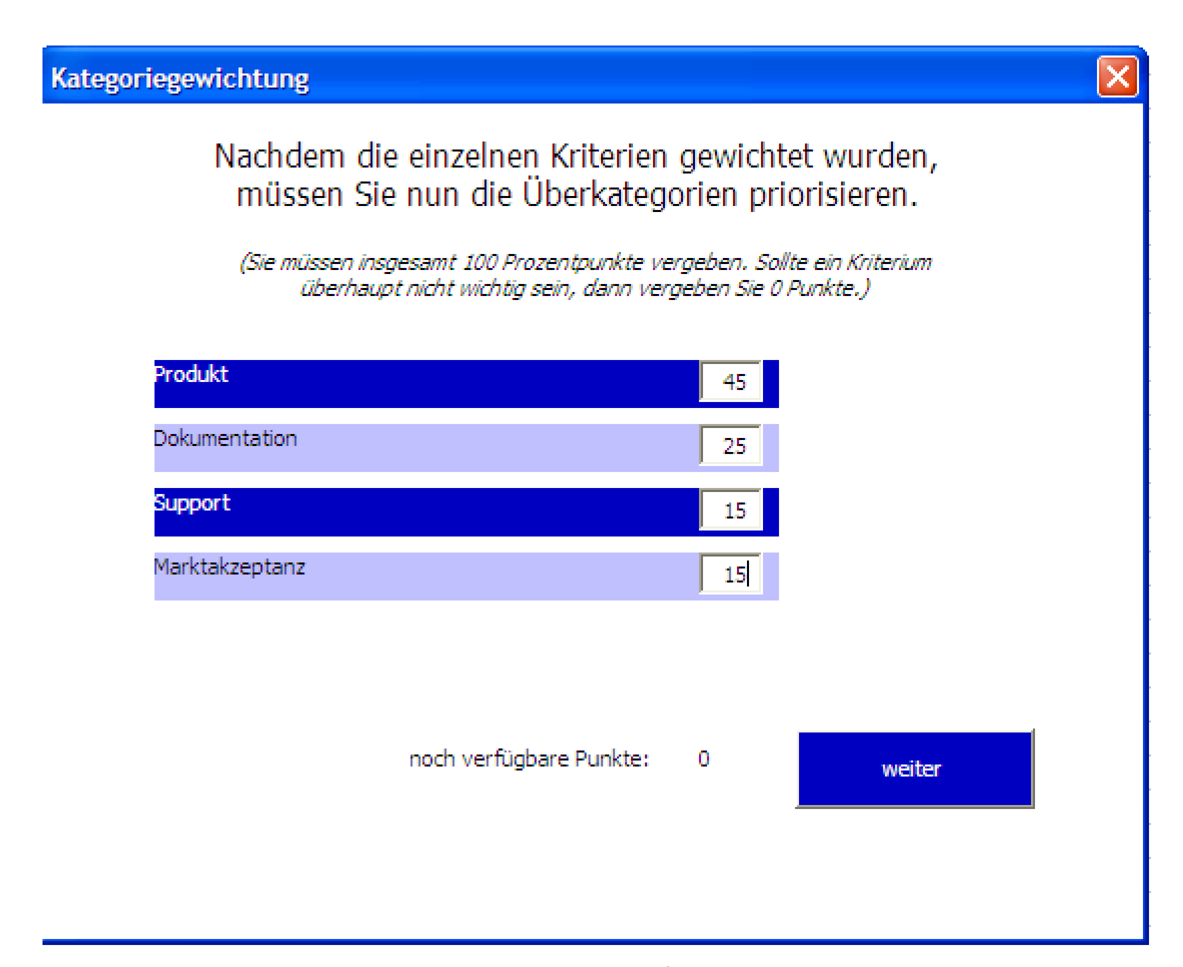


Abb. 20: Prozessschritt 3: Priorisierung festlegen (5/5)

Anhang 17: Screenshots

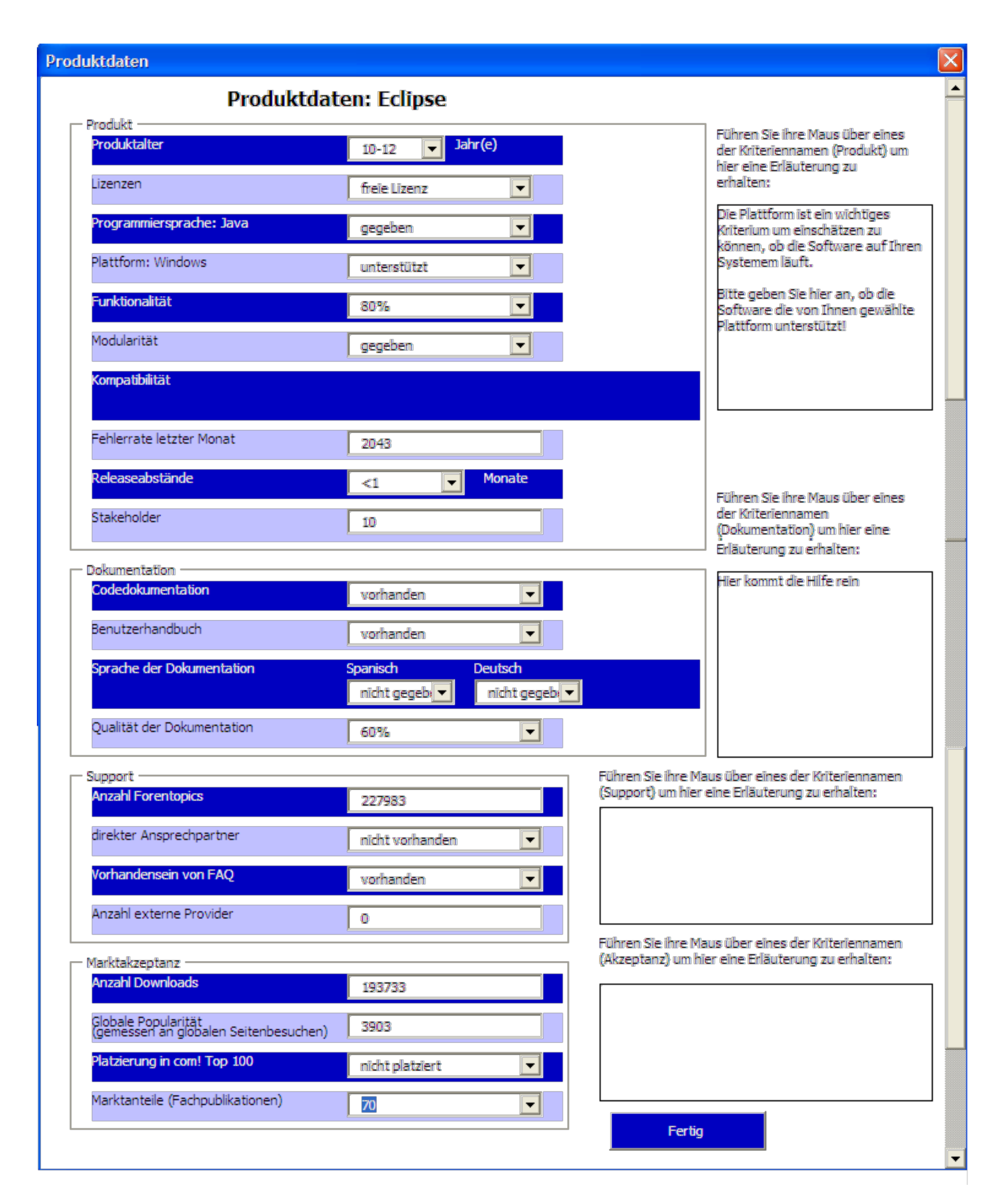


Abb. 21: Prozessschritt 4: Datengenerierung für Projekt

Anhang 18: Screenshots

	A	B	C	D	E	F	G	H
1	Kriterien		Gewichtung in %		Bewertung basierend		Verrechnung Bewertung *	
2			Oberkat.	Unterkat.	auf Produktfakten	Produktfakten	Gewichtung	
3	Produkt		45				0.5355	
4		Produktalter		15	1	12. Okt		0.15
5		Lizenzen		20	1	freie Lizenz		0.2
6		Programmiersprache: Java		2	2	gegeben		0.04
7		Plattform: Windows		10	2	unterstützt		0.2
8		Funktionalität		30	1.5	80%		0.45
9		Modularität		0	2	gegeben		0
10		Kompatibilität		0				0
11		Fehlerrate		8	0	2043		0
12		Releaseabstände		0	2	<1		0
13		Stakeholder		15	1	10		0.15
14	Dokumentation		25				0.3375	
15		Codedokumentation		35	2	vorhanden		0.7
16		Benutzerhandbuch		10	2	vorhanden		0.2
17		Sprache der Dokumentation		10	0	nicht gegeben;nicht gegeben		0
18		Qualität der Dokumentation		45	1	60%		0.45
19								
20	Support		15				0.189	
21		Anzahl Forenbeiträge letzter Monat		21	2	227983		0.42
22		Direkter Ansprechpartner		12	0	nicht vorhanden		0
23		Vorhandensein von FAQ		42	2	vorhanden		0.84
24		Anzahl externe Provider		25	0	0		0
25								
26	Marktakzeptanz		15				0.225	
27		Anzahl Downloads		20	2	193733		0.4
28		Globale Popularität (gemessen an globalen Seitenbesuchen)		17	2	3903		0.34
29		Platzierung in com! Top 100 der Open Source Projekte		25	0	nicht platziert		0
30		Marktanteile (auf Basis von Fachpublikationen)		38	2	70		0.76
31								
32	Gesamtsumme							1.287

Abb. 22: Prozessschritt 5: Reifegradermittlung

Anhang 19: Screenshots

Kriterien		Gewichtung in % Oberkat.	Unterkat.	Eclipse Bewertung basierend auf Produktfakten	Verrechnung Bewertung *	NetBeans Bewertung basierend auf	Verrechnung Bewertung *
Produkt		45			0.5355		0.4815
	Produktalter		15	1	0.15	1	0.15
	Lizenzen		20	1	0.2	1	0.2
	Programmiersprache: Java		2	2	0.04	2	0.04
	Plattform: Windows		10	2	0.2	0	0
	Funktionalität		30	1.5	0.45	1.5	0.45
	Modularität		0	2	0	2	0
	Kompatibilität		0		0		0
	Fehlerrate		8	0	0	1	0.08
	Releaseabstände		0	2	0	1	0
	Stakeholder		15	1	0.15	1	0.15
Dokumentation		25			0.3375		0.23125
	Codedokumentation		35	2	0.7	2	0.7
	Benutzerhandbuch		10	2	0.2	0	0
	Sprache der Dokumentation		10	0	0	0	0
	Qualität der Dokumentation		45	1	0.45	0.5	0.225
Support		15			0.189		0.2625
	Anzahl Forenbeiträge letzter Monat		21	2	0.42	2	0.42
	Direkter Ansprechpartner		12	0	0	2	0.24
	Vorhandensein von FAQ		42	2	0.84	2	0.84
	Anzahl externe Provider		25	0	0	1	0.25
Marktakzeptanz		15			0.225		0.1545
	Anzahl Downloads		20	2	0.4	2	0.4
	Globale Popularität (gemessen an globalen Seitenbesuchen)		17	2	0.34		0
	Platzierung in com! Top 100 der Open Source Projekte		25	0	0	1	0.25
	Marktanteile (auf Basis von Fachpublikationen)		38	2	0.76	1	0.38
Gesamtsumme					1.287		1.12975

Abb. 23: Prozessschritt 6: Selektion

Quellenverzeichnis

Literaturverzeichnis

Asche, M. / Bauhus, W. / Mitschke, E. / Seel, B. (2008)	Open Source: Kommerzialisierungsmöglichkeiten und Chancen für die Zusammenarbeit von Hochschulen und Unternehmen, Band 3, Münster: Waxmann Verlag
Russo, B. / Sillitti, A. (2010)	Agile Technologies in Open Source Development, Hershey: IGI Global
Haaland, K. / Groven , A. / Glott, R. / Tannenberg, A. (2009)	Free/Libre Open Source Quality Models - a comparison between two approaches, Uni Jena

Intranet- und Internetquellen

Heise (2012)	http://www.heise.de/open/meldung/Gartner-Open-Source-ist-ueberall-217214.html, Abruf: 05.06.2012
o. V. (2012)	http://www.cio.de/subnet/oracle-finance/2235636/index4.html, Abruf: 05.06.2012
Heise (2012)	http://www.heise.de/open/meldung/Gartner-Open-Source-ist-ueberall-217214.html, Abruf: 22.06.2012
Wikipedia (2012)	http://en.wikipedia.org/wiki/File:QSOS-processus-en.png, Abruf: 22.06.2012
Wikipedia (2012)	http://en.wikipedia.org/wiki/File:OMMStructure.png, Abruf: 22.06.2012
Wikipedia (2012)	http://en.wikipedia.org/wiki/OpenSource_Maturity_Model, Abruf: 30.06.2012
o. V. (2012)	http://bolsa.info.unlp.edu.ar/campamento/campamento/documentos/GB_Expert_Letter_Open_Source_Maturity_Model_1.5.3.pdf, Abruf: 30.06.2012
o. V. (2012)	www.nrcfoss.au-kbc.org.in/maturity, Abruf: 30.06.2012
o. V. (2012)	http://www.sita.co.za/FOSS/Gov_Malaysia-OSS_Guide.pdf , Abruf: 22.06.2012
o. V. (2012)	http://www.laum.uni-hannover.de/ilr/lehre/Ptm/Ptm_BewNwa.htm, Abruf: 05.06.2012

Quellen für die Analyse der Projekte

Abrufdatum für alle angegebenen Quellen: 1.Juli 2012

Open Office:

Produktalter http://de.wikipedia.org/wiki/OpenOffice.org

Downloadzahl http://www.chip.de/downloads/OpenOffice_13004346.html

Alexa http://www.alexa.com/siteinfo/openoffice.org#

Fehlerrate https://issues.apache.org/ooo/buglist.cgi?chfieldfrom=2012-06-01&chfieldto=Now&query_ format=advanced&resolution=---&order=priority%2Cbug_severity&limit=0

Release http://de.wikipedia.org/wiki/OpenOffice.org

Forumeinträge http://de.openoffice.info/

Sponsoren http://www.openoffice.org/de/sponsoren/index.html

Marktanteil http://de.wikipedia.org/wiki/Apache_OpenOffice

7zip:

Produktalter http://de.wikipedia.org/wiki/7-Zip

Downloadzahl http://www.chip.de/downloads/7-Zip-32-Bit_13004776.html

Alexa http://www.alexa.com/siteinfo/7-zip.org#

Fehlerrate http://sourceforge.net/tracker/?limit=100&func=&group_id=14481&atid=114481&assignee=&status=&category=&artgroup=&keyword=&submitter=&artifact_id=&assignee=&status=&category=&artgroup=&submitter=&keyword=&artifact_id=&submit=Filter

Release http://de.wikipedia.org/wiki/7zip

Forumbeiträge http://sourceforge.net/tracker/?group_id=14481&atid=114481Joomla

Joomla:

Produktalter http://www.joomla.de/entdecken/geschichte.html

Downloadzahl http://www.chip.de/downloads/Joomla_21687904.html

Alexa http://www.alexa.com/siteinfo/joomla.org#

Fehlermeldungen http://joomlacode.org/gf/project/joomla/tracker/? ac-tion=TrackerItemBrowse&tracker_id=32&start=3875

Release http://de.wikipedia.org/wiki/Joomla

Forum http://www.joomlaportal.de/forum.php

Marktanteil http://www.webkalkulator.com/cmsvergleich.asp

Firefox:

Produktalter http://de.wikipedia.org/wiki/Joomla

Downloadzahl http://www.chip.de/downloads/Firefox_13014344.html

Alexa http://www.alexa.com/siteinfo/mozilla.org#

Fehlermeldungen https://bugzilla.mozilla.org/buglist.cgi?resolution=---;chfieldto=Now;query_format=advanced;chfieldfrom=2012-06-01;bug_status=NEW

Release http://de.wikipedia.org/wiki/Firefox

Forumbeiträge http://www.camp-firefox.de/forum/viewforum.php?f=1&start=0

Marktanteil http://www.browser-statistik.de/statistiken/

Apache:

Produktalter http://de.wikipedia.org/wiki/Apache_HTTP_Server

Downloadzahl http://www.chip.de/downloads/Apache-HTTP-Server_22022661.html

Alexa http://www.alexa.com/siteinfo/apache.org#

Fehlermeldungen https://issues.apache.org/bugzilla/buglist.cgi?query_format=specific&order=relevance+desc&bug_status=__all__&product=&content=2012

Release http://de.wikipedia.org/wiki/Apache_HTTP_Server

Sponsoren http://www.apache.org/foundation/thanks.html

Forumeinträge http://www.apachefriends.org/f/

Marktanteil http://www.heise.de/open/meldung/Apache-gewinnt-Marktanteile-hinzu-140144.html

JBoss Application Server:

Produktalter http://de.wikipedia.org/wiki/JBoss

Alexa http://www.alexa.com/siteinfo/jboss.org#

Release https://issues.jboss.org/browse/AS7#selectedTab=com.atlassian.jira.ext. calen-dar%3Aissuecalendar-panel

Fehlermeldugnen https://issues.jboss.org/secure/IssueNavigator.jspa?

Forumeinträge https://community.jboss.org/en/jbossas7?view=discussions

Gimp:

Downloadzahl http://www.chip.de/downloads/GIMP-32-Bit_12992070.html

Produktalter http://de.wikipedia.org/wiki/GIMP

Alexa http://www.alexa.com/siteinfo/gimp.org#

Release http://de.wikipedia.org/wiki/Gimp

Sponsoren http://www.gimp.org/donating/sponsors.html

Forumeinträge http://www.gimpforum.de/

Piwik:

Produktalter http://de.wikipedia.org/wiki/Piwik

Downloadzahl http://www.chip.de/downloads/Piwik_47669638.html

Alexa http://www.alexa.com/siteinfo/piwik.org#

Sponsoren http://piwik.org/about/sponsors/

Forumeinträge http://forum.piwik.org/

Marktanteil http://piwikblog.de/category/piwik/

Eclipse:

Downloadzahl http://www.chip.de/downloads/Eclipse_24547633.html

Produktalter http://de.wikipedia.org/wiki/Eclipse_%28IDE%29
Alexa http://www.alexa.com/siteinfo/eclipse.org#
Fehlermeldungen https://bugs.eclipse.org/bugs/buglist.cgi?chfieldto=Now;query_format=advanced;chfieldfrom=2012-06-01;classification=Eclipse
Release http://de.wikipedia.org/wiki/Eclipse_(IDE)
Sponsoren http://www.eclipse.org/corporate_sponsors/
Forumeinträge http://www.eclipse.org/forums/
Marktanteil http://www.heise.de/developer/meldung/10-Jahre-Eclipse-Konsolidierung-des-Java-IDE-Markts-1370644.html

Phorum:

Produktalter http://de.wikipedia.org/wiki/Phorum
Downloadanzahl http://www.chip.de/downloads/Phorum_29988494.html
Release http://de.wikipedia.org/wiki/Phorum
Alexa http://www.alexa.com/siteinfo/phorum.org#
Forumeinträge http://www.eclipse.org/forums/

Typo3:

Produktalter http://de.wikipedia.org/wiki/TYPO3
Downloadzahl http://www.chip.de/downloads/Typo3_24619810.html
Alexa http://www.alexa.com/siteinfo/typo3.org#
Fehlermeldungen http://bugs.typo3.org/view_all_bug_page.php?page_number=2
Release http://de.wikipedia.org/wiki/Typo3
Sponsoren http://typo3.org/search/?q=stakeholder&id=180&L=0
Forumeinträge http://www.typo3forum.net/forum/forum.php
Marktanteil http://www.webkalkulator.com/cmsvergleich.asp

Zimbra:

Downloadzahl http://www.chip.de/downloads/Zimbra-Desktop_32360234.html
Produktalter http://www.heise.de/open/meldung/Exchange-Alternative-Zimbra-mit-neuen-Features-140089.html
Alexa http://www.alexa.com/siteinfo/zimbra.com#
Release http://de.wikipedia.org/wiki/Zimbra
Sponsoren http://de.wikipedia.org/wiki/Zimbra
Forumeintrag http://www.zimbra.com/forums/forum.php

Asterisk:

Downloadzahl http://www.chip.de/downloads/Asterisk-Win32_21745063.html
Fehlermeldung https://issues.asterisk.org/view_all_bug_page.php
Alexa http://www.alexa.com/siteinfo/asterisk.org#
Release http://de.wikipedia.org/wiki/Asterisk_%28Telefonanlage%29

Sponsoren http://www.asterisk.org/

<u>K Meleon:</u>

Produktalter http://de.wikipedia.org/wiki/K-Meleon

Downloadzahl http://www.chip.de/downloads/K-Meleon_13011154.html

Fehlerrate http://kmeleon.sourceforge.net/bugs/findbug.php?summary=&description=&project%5B%5D=1&creator=&daterange=&moddays=200&submit=Search

Release http://de.wikipedia.org/wiki/K-Meleon

Sponsoren http://kmeleon.sourceforge.net/

Forumbeiträge http://kmeleon.sourceforge.net/forum/

Marktanteil http://www.browser-statistik.de/statistiken/

<u>Netbeans:</u>

Produktalter http://de.wikipedia.org/wiki/NetBeans_IDE

Downloadzahl http://www.chip.de/downloads/NetBeans_29282799.html

Alexa http://www.alexa.com/siteinfo/netbeans.org#

Release http://de.wikipedia.org/wiki/Netbeans

Sponsoren http://netbeans.org/community/partners/index.html

Forumeinträge http://www.netbeans-forum.de/index.php?sid=2f9d60aed58fdca247986ff90888b386

Marktanteil http://www.webwork-magazin.net/netbeans-6-9-freigegeben/2630

<u>Grails:</u>

Downloadzahl http://www.chip.de/downloads/Grails_34433310.html

Alexa http://www.alexa.com/siteinfo/grails.org#

Fehlerrate http://jira.grails.org/secure/IssueNavigator.jspa?pager/start=50

Release http://de.wikipedia.org/wiki/Grails

Sponsoren http://www.springsource.com/groovy-grails-consulting

Forumeinträge http://grails.1312388.n4.nabble.com/

<u>Arch Linux:</u>

Produktalter http://de.wikipedia.org/wiki/Arch_Linux

Alexa http://www.alexa.com/siteinfo/archlinux.org#

Fehlerrate https://bugs.archlinux.org/index.php?string=&project=1&type%5B%5D=&sev%5B%5D=&pri%5B%5D=&due%5B%5D=&reported%5B%5D=&cat%5B%5D=&status%5B%5D=open&percent%5B%5D=&opened=&dev=&closed=&duedatefrom=&duedateto=&changedfrom=&changedto=&openedfrom=2012-06-01&openedto=&closedfrom=&closedto=&do=index

Forumeinträge https://bbs.archlinux.org/

<u>Pidgin:</u>

Downloadzahl http://www.chip.de/downloads/Pidgin_13009923.html

Produkalter http://de.wikipedia.org/wiki/Pidgin_%28Instant_Messenger%29

Alexa http://www.alexa.com/siteinfo/pidgin.im#

Fehlerrate http://developer.pidgin.im/report/3?USER=anonymous&page=20

Forumeinträge http://pidgin-im.de/

<u>Sunbird:</u>

Downloadzahl http://www.chip.de/downloads/Sunbird_13014098.html

Produktalter http://de.wikipedia.org/wiki/Mozilla_Sunbird

Alexa http://www.alexa.com/siteinfo/sunbird-kalender.de#

Release http://de.wikipedia.org/wiki/Mozilla_Sunbird

Sponsoren http://de.wikipedia.org/wiki/Mozilla_Sunbird

Forumeinträge http://www.sunbird-kalender.de/forum/

<u>mySQL:</u>

Produktalter http://de.wikipedia.org/wiki/MySQL

Downloadanzahl http://www.chip.de/downloads/MySQL_13000125.html

Alexa http://www.alexa.com/siteinfo/mysql.de#

Fehlerrate http://bugs.mysql.com/search.php?search_for=&status=Active&severity=&limit =All&order_by=&cmd=display&direction=ASC&bug_type=&os=0&phpver=&bug_age=30

Release http://de.wikipedia.org/wiki/Mysql

Sponsoren http://www.mysql.de/partners/

Forumeinträge http://forums.mysql.com/

Marktanteil http://www.mysql.de/why-mysql/marketshare/



Fakultät Wirtschaft

Der Dualen Hochschule Baden-Württemberg Stuttgart

Projekt im Modul Anwendung von Konzepten des Geschäftsprozessmanagement im 5. Semester

Leitfaden zur Einführung von Open Source Software in Organisationen

Autoren: Ralf Hecktor, Daniel Hohmann, Madeline Klink, Jana Petrovic, Timo Pfister, Gerd Radecke

Studiengang: Wirtschaftsinformatik – International Business Information Management

Aufgabenstellung/Betreuung: Prof. Dr. Thomas Kessel

Datum: 16.01.2012

Inhaltsverzeichnis

1 Einleitung 1

2 Orientierungsphase 4

2.1 Motivationen und Herausforderungen bei der Einführung von OSS 4

2.1.1 Kostenreduktion 4

2.1.2 Höherer Erfüllungsgrad der Anforderungen an die Funktionalität 5

2.1.3 Datensicherheit für die Organisation 6

2.1.4 Planungssicherheit für die Organisation 7

2.1.5 Einhalten rechtlicher Vorgaben 8

2.1.6 Erzeugen positiver Außenwirkung 9

2.1.7 Erzeugen einer positiven Innenwirkung 10

2.1.8 Gewährleistung und Verbesserung der Kommunikation mit Partnern 10

2.2 Rechtliche Rahmenbedingungen 11

2.2.1 Der Lizenz-Begriff und das deutsche Urheberrecht 12

2.2.2 Lizenzarten 12

2.2.3 Urheberrechtliche und vertragsrechtliche Fragen 17

2.2.4 Nutzung von Open Source Software ohne Modifikationsabsichten 18

2.2.5 Checkliste 18

2.3 Was die folgenden Kapitel des Ratgebers leisten 20

3 Evaluationsphase 23

3.1 Auswahl der passenden Software 23

3.1.1 Funktionale Anforderungen 24

3.1.2 Nicht funktionale Anforderungen 25

3.1.3 Methodik zur Auswahl passender Software 27

3.1.4 Anpassung der Software an die Anforderungen 28

3.1.5 Auswahltests 29

3.1.6 Anwendertests 30

3.1.7 Checkliste 31

3.2 Qualität und Reifegrad 33

3.2.1 Bewertung der Community ... 33
3.2.2 Bewertung der Softwarequalität ... 36
3.2.3 Checkliste ... 38
3.3 Betrachtung der Total Cost of Ownership ... 39
3.3.1 Einführung in TCO ... 39
3.3.2 Zweck von TCO Modellen ... 40
3.3.3 TCO Modell nach Gartner ... 41
3.3.4 Maßnahmen für die Erhebung der TCO im Zuge einer Softwaremigration ... 45
4 Transformationsphase ... 50
4.1 Migration und Rollout ... 50
4.1.1 Definition ... 50
4.1.2 Migrationsarten ... 52
4.1.3 Migrationsstrategien ... 54
4.1.4 Migrationsvorbereitungen ... 55
4.1.5 Migrationsansätze ... 59
4.1.6 Umstellungs- und Übergabestrategien ... 61
4.1.7 Besonderheiten bei Open Source Software ... 63
4.1.8 Checkliste ... 64
4.2 Anwenderschulung und Dokumentation ... 67
4.2.1 Schulung durch einen externen Anbieter ... 68
4.2.2 Schulung durch interne Mitarbeiter ... 70
4.2.3 Kontrolle und Feedback ... 71
4.2.4 Dokumentation ... 73
4.2.5 Besonderheiten bei OSS als Schulungsgegenstand ... 74
4.2.6 Checkliste ... 74
5 Produktivphase ... 77
5.1 Interaktion mit der Community ... 77
5.1.1 Zeitpunkt der Interaktion ... 77
5.1.2 Zweck der Interaktion ... 78
5.1.3 Definition der Kommunikationswege ... 79

5.1.4 Checkliste 81

6 Fazit: Open Source Software - ja oder nein? 84

7 Quellen 88

Abbildungsverzeichnis

Abb. 1: Verwendung der Lizenzkosten 5

Abb. 2: Basisfaktoren der TCO nach Gartner 42

Abb. 3: Mögliche Migrationspfade und Alternativen 52

Abb. 4: Punktumstellung 61

Abb. 5: Paketumstellung mit Komponentenersatz 62

Abb. 6: Paketumstellung mit Paralleleinsatz 63

Tabellenverzeichnis

Tabelle 1: Entscheidungsmatrix 22

Tabelle 2: Anforderungserfüllung mit Gewichtung 28

Tabelle 3: Gegenüberstellung der Kostenerhebung für proprietäre und Open Source Lösungen 48

Tabelle 4: Beispielkalkulation der TCO 49

Tabelle 5: Softwarebewertungsskala 57

Tabelle 6: Schulungsarten 69

Tabelle 7: Entscheidungsmatrix ausgefüllt 86

1 Einleitung

Mit der zunehmenden Digitalisierung nahezu aller Prozesse, sowohl in der Arbeitswelt als auch im Privatleben, geht ein stetig wachsender Bedarf für den Einsatz von Software einher. War zu Beginn des Zeitalters der Maschinenlesbarkeit noch ein Großteil der verfügbaren Software an die jeweilige Hardware gebunden, so etablierte sich zu Beginn der 1970er Jahre das Konzept von der Hardware gelöster Softwarepakete. Dieses Prinzip hat mit der voranschreitenden Verbreitung von Personal-Computern (PCs) bis heute stetig an Bedeutung gewonnen.

Es lässt sich hierbei zwischen proprietärer, meist kommerzieller Software mit nicht frei zugänglichem Quellcode (Closed Source Software - CSS) und quelloffener, kostenloser, frei zu verbreitender und modifizierbarer Software (Open Source Software - OSS) unterscheiden.

Heute kommt in den allermeisten Organisationen vorwiegend proprietäre Software, also CSS, zum Einsatz. Jedoch rückt mit zunehmendem Kostendruck in der IT und der Verbreitung offener Standards OSS immer mehr in den Fokus der Betrachtung.

Diese Arbeit stellt einen (dem OSS-Gedanken folgend) kostenlosen und frei zu verbreitenden Ratgeber dar. Dieser soll dazu dienen, die Einführung von OSS in Organisationen zu unterstützen. Er richtet sich dabei vorrangig an kleine und mittelständische Unternehmen, die OSS (unverändert oder modifiziert) zur softwareseitigen Unterstützung Ihrer Arbeitsprozesse verwenden möchten. Ein Großteil der Überlegungen, die zur Verwertung von OSS als Geschäftsgrundlage nötig wären, werden hier nicht abgedeckt.

Dieses Dokument adressiert in erster Linie Entscheider in der Informationstechnologie und soll dem Leser einen Leitfaden zur zielführenden Vorgehensweise bei der Evaluation und möglichen Einführung von OSS an die Hand geben. Es trägt hierzu die Erkenntnisse bereits veröffentlichter Arbeiten zu diesem Thema zusammen und gibt Empfehlungen zur weiterführenden Recherche. Der Ratgeber erhebt dabei stets den Anspruch, möglichst neutral zu beleuchten, unter welchen Umständen sich der Einsatz von OSS tatsächlich lohnt. Ziel ist es, den Leser darin kompetent zu machen, selbstständig zu entscheiden, ob und in welcher Form er OSS in seiner Organisation nutzen möchte.

Der Leitfaden gliedert sich im Groben in vier Phasen:

- die *Orientierungsphase,* in der sondiert wird, ob die Einführung von OSS für die eigene Organisation grundsätzlich in Frage kommt bzw. sinnhaft ist (beleuchtet werden Motivationen und Herausforderungen sowie rechtliche Rahmenbedingungen)
- die *Evaluationsphase,* in der konkrete OSS-Lösungen hinsichtlich ihrer Funktionalitäten, ihres Qualitätsniveaus und der mit ihnen verbundenen Kostenstruktur analysiert werden, um die Entscheidung für ein Produkt zu ermöglichen
- die *Transformationsphase*, in der nach der Entscheidung für ein Produkt der konkrete Rollout dieser Lösung geplant und durchgeführt wird (betrachtet werden die technischen Handlungsschritte, aber auch die Dokumentation sowie die Schulung der Anwender)
- die *Produktivphase* nach der Einführung, in der fortlaufend das reibungslose Funktionieren der Software sichergestellt werden muss (beleuchtet wird v.a. die Sicherstellung von Support)

Innerhalb der einzelnen Kapitel wird zunächst ein kurzer Einblick in die vorhandene Theorie zur jeweiligen Thematik gegeben, diese mit eigenen Gedanken angereichert und hieraus praktische Empfehlungen abgeleitet. Diese praktischen Empfehlungen werden anschließend in einer Checkliste oder einer ausfüllbaren Evaluationsmatrix manifestiert. Zu Illustrationszwecken wird auf diese ein fiktives Praxisbeispiel angewendet:

> *Bei unserer Beispiel-Organisation handelt es sich um ein 250 Mitarbeiter starkes, mittelständisches Unternehmen aus der Dienstleistungsbranche mit 45 Millionen Euro Jahresumsatz. Jeder Mitarbeiter hat einen PC-Arbeitsplatz und korrespondiert mit Kunden und Kollegen vor allem telefonisch und per E-Mail. Das Unternehmen beherbergt eine kleine interne IT-Abteilung mit einer Vollzeitkraft. Die Organisation erwägt, ihr gegenwärtig verwendetes proprietäres Groupware- und Nachrichtensystem (E-Mail-Client mit gemeinsamer Kalenderfunktion sowie dahinter stehende Serverapplikationen) durch eine OSS-Lösung zu ersetzen. Die Hauptmotivationen hierfür sind die Aussicht auf Kostenersparnisse und die Herauslösung aus der Abhängigkeit vom Anbieter der aktuell verwendeten Software.*

Die bereitgestellten Checklisten sind als unterstützende Schablonen zu verstehen, die je nach Belieben und individuellen Bedürfnissen modifiziert werden dürfen. Das fiktive Praxisbeispiel soll den Leser dabei unterstützen, diese Schablonen auf seine jeweilige Organisation anzuwenden.

Abhängig von den jeweiligen Gegebenheiten wird der Leser über diesen Ratgeber hinausgehende Literatur zu Rate ziehen wollen. Daher wird am Ende jedes Kapitels noch einmal explizit auf empfehlenswerte vertiefende Literatur hingewiesen.

2 Orientierungsphase

Im Entscheidungsprozess, ob und welche OSS in der eigenen Organisation eingeführt werden soll, steht an erster Stelle die Überlegung, ob OSS prinzipiell überhaupt als mögliche Lösung in Frage kommt und weiterführende Evaluationen Sinn ergeben. Um diese Frage beantworten zu können, gilt es zunächst zu ermitteln, welche Umstände zur Nutzung von OSS motivieren und welche der Einführung im Wege stehen könnten. Außerdem sollte man sich im Klaren über die rechtlichen Rahmenbedingungen der Nutzung von OSS sein. Im folgenden Kapitel sollen das relevante Wissen vermittelt und hilfreiche Denkanstöße gegeben werden, um diese Grundsatzentscheidung fällen zu können.

2.1 Motivationen und Herausforderungen bei der Einführung von OSS

Zunächst sollen einige grundsätzliche Motivationen aufgezeigt werden, OSS in der eigenen Organisation zu verwenden und diese möglicherweise sogar weiterzuentwickeln. Zu jeder der genannten Motivationen werden die zu Grunde liegende Chancen, aber auch mit selbigen verbundene Risiken genannt. Aus diesen Risiken werden dann Herausforderungen abgeleitet, denen sich die Organisation im Zuge der Einführung von OSS möglicher Weise stellen muss.

2.1.1 Kostenreduktion

Die wohl prominenteste und augenscheinlichste Motivation, OSS zu verwenden ist die der Kostenreduktion. Die Chance liegt hier in den nicht zu entrichtenden Lizenzkosten für die Anschaffung und Nutzung der Software. Das Geld, das bei den entfallenden Lizenzkosten gespart wird, kann dann an anderer Stelle innerhalb der Organisation investiert werden (z.B. in Forschung, Mitarbeiterschulung oder Öffentlichkeitsarbeit). Abbildung 1 soll diese Überlegung verdeutlichen:

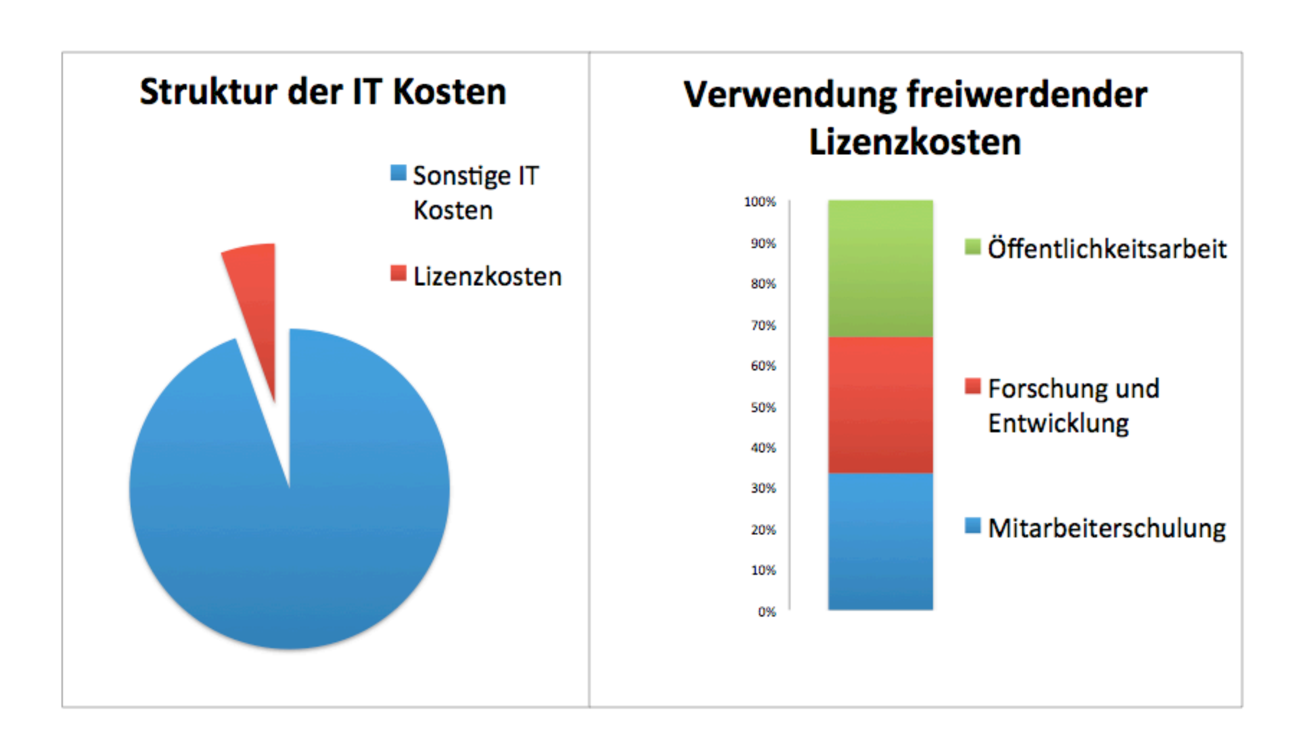


Abb. 1: Verwendung der Lizenzkosten

Man sollte jedoch nicht aus den Augen verlieren, dass die Lizenzkosten nicht der einzige zu berücksichtigende Kostenfaktor sind. Kosten beispielsweise für Wartung, Support, Anwenderschulung und Modifikationen an der Software werden sich aller Wahrscheinlichkeit nach anders entwickeln als bei einer proprietären (abzulösenden) Software. Diese Kosten können höher, aber auch geringer ausfallen als bei proprietärer Software. Auch die Kosten für die Evaluation selbst sollten nicht außer Acht gelassen werden. Die Herausforderung, die sich hieraus ableiten lässt, ist die Notwendigkeit, die Gesamtkosten der Nutzung einer OSS zu ermitteln und angemessen zu managen. Ausführliche Überlegungen zu diesem Thema sollen in Kapitel *3.3 Betrachtung der Total Cost of Ownership* angestellt werden.

2.1.2 Höherer Erfüllungsgrad der Anforderungen an die Funktionalität

Eine weitere Motivation, OSS einzuführen besteht in der Hoffnung, dass diese die Anforderungen der Organisation besser abdeckt. Eine der Chancen, die zu dieser Hoffnung Anlass geben, besteht in der Tatsache, dass in OSS-Projekten die Entwickler in vielen Fällen gleichzeitig Benutzer der Software sind. Somit verstehen sie die Anwenderbedürfnisse sehr viel besser als Entwickler proprietärer Software. Dies ist häufig vor allem im (natur-) wissenschaftli-

chen Umfeld der Fall, beispielsweise bei der mathematischen Testumgebung *FreeMat*[1] oder der Planeten-Simulationssoftware *NASA World Wind*[2]. Eine weitere wichtige Chance besteht in der Offenheit des Quellcodes. Diese bietet die Möglichkeit, die Software eigenständig oder durch Dritte den Anforderungen der Organisation entsprechend zu modifizieren. Ein bekanntes Beispiel hierfür stellt *LibreOffice*[3] dar, das als Reaktion auf die mangelnde Aktivität und Kommunikationsbereitschaft seitens Oracle im Projekt *OpenOffice* gegründet wurde, und einen Großteil des Codes dieser OSS wiederverwendet. Dem neu entstandenen *LibreOffice* sicherten in diesem Zusammenhang neue Unternehmenspartner (Google, Red Hat, Novell) Unterstützung zu[4].

Zudem kann die Organisation möglicherweise (z.B. durch das Bereitstellen von Ressourcen) derart Einfluss auf die hinter dem jeweiligen OSS-Projekt stehenden Community nehmen, dass diese die Software in die gewünschte Richtung weiterentwickelt. Ein prominentes Beispiel hierfür bietet das Java-Toolkit *Dojo*[5], mit dem IBM eine enge Partnerschaft pflegt. Auf den Aspekt der Funktionalitätsanforderungen wird in Kapitel *3.1. Auswahl der passenden Software* eingegangen, auf die Frage nach einer angemessenen Zusammenarbeit mit der jeweiligen Community in Kapitel *5.1. Interaktion mit der Community.*

2.1.3 Datensicherheit für die Organisation

Eine weitere wichtige Motivation für Organisationen (insbesondere für Unternehmen) liegt in der Gewährleistung der Sicherheit ihrer Daten. Mit der Einführung von OSS ist oftmals die Sorge verbunden, dass diese aufgrund der Offenheit des Quellcodes unsicherer sein müsse. Tatsächlich kann durch die freie Zugänglichkeit der Code problemlos von Kriminellen eingesehen werden. Allerdings bringt OSS zugleich die Chance mit sich, dass (ebenfalls ermöglicht durch die Offenheit des Codes) die Einsichtnahme einer Vielzahl an Entwicklern und Testern dafür sorgt, Sicherheitslücken in der Software zeitnah erkennen und schließen zu können. In

---

[1] http://freemat.sourceforge.net/

[2] http://worldwind.arc.nasa.gov/java/

[3] http://de.libreoffice.org/

[4] http://www.heise.de/newsticker/meldung/OpenOffice-wird-zu-LibreOffice-Die-OpenOffice Community-loest-sich-von-Oracle-1097356.html

[5] http://www.dojotoolkit.org/

aller Regel überwiegt der zuletzt genannte Effekt; Sicherheitslücken werden sehr schnell nach ihrer Entdeckung geschlossen. Die Angst vor Angriffen durch Kriminelle ist daher nicht unbegründet, sollte aber im Einzelfall kritisch überdacht werden. Von entscheidender Wichtigkeit ist die Reife der Community, die hinter der jeweiligen OSS steht (und damit die Geschwindigkeit, mit der Sicherheitslücken geschlossen werden). Die Sicherheitsaspekte von OSS werden (neben anderen Themen) in Kapitel *3.2 Qualität und Reifegrad* genauer erläutert.

2.1.4 Planungssicherheit für die Organisation

Die Gewährleistung der Sicherheit für die Organisation ist ein wichtiger Motivationsfaktor, jedoch nicht allein hinsichtlich der Datensicherheit, sondern auch im Sinne von Planungssicherheit.

Eine Chance bei der Einführung von OSS stellt in diesem Zusammenhang die Unabhängigkeit von einem Anbieter proprietärer Software dar. Abhängig von einem solchen Lieferanten zu sein, bringt für eine Organisation stets eine gewisse Unsicherheit mit sich. Im Falle eines kleinen Anbieters maßgeschneiderter Software besteht die Möglichkeit, dass dieser insolvent wird und das einmal verkaufte Produkt nicht mehr weiterentwickelt und keinen Support mehr dafür leisten kann. Wenn sich beispielsweise eine mittelständische Wäscherei ihr Rechnungssystem von einer kleinen Softwareschmiede kommerziell entwickeln lässt, ist sie stark abhängig vom Überleben dieses Zulieferers. Im Falle eines größeren Lieferanten (dessen ökonomische Standfestigkeit tendenziell höher einzuschätzen ist) liegt die Gefahr vielmehr darin, dass dieser sich entweder entschließt, das von der eigenen Organisation verwendete Produkt in eine nicht erwünschte Richtung weiterzuentwickeln oder es sogar ganz aus seiner Produktpalette zu eliminieren. In beiden Fällen ist der Quellcode der Software für die eigene Organisation nicht einsehbar und damit eine Weiterentwicklung in Eigenregie ausgeschlossen.

Unter der Verwendung von OSS kann eine gewisse Planungssicherheit dadurch gewährleistet werden, dass (auf Grund der freien Einsehbarkeit des Codes) prinzipiell zu jedem Zeitpunkt die Möglichkeit besteht, die Weiterentwicklung selbst in die Hand zu nehmen oder einem Dritten (gegen Entlohnung) anzuvertrauen. Eine weitere Chance in puncto Planungssicherheit stellt die Tatsache dar, dass in Open Source-Projekten auch der Entwicklungsprozess sehr transparent gemacht wird. Es ist dadurch recht wahrscheinlich, dass die mittelfristige

Roadmap für das Produkt frei einsehbar ist. Ein gutes Beispiel hierfür stellt der *Java Community Process*[6] dar. Solange um die verwendete OSS eine aktive Community besteht oder ein Unternehmen das Projekt federführend weiterentwickelt (Beispiel *Dojo*), sind Anpassungen in Eigenregie möglicher Weise noch nicht einmal nötig. Bei regem Austausch und angemessener Kommunikation mit der Community ist es durchaus denkbar, diese in eine von der Organisation gewünschte Richtung zu lenken. Ein gewisses Risiko in diesem Zusammenhang besteht darin, dass eine zum Zeitpunkt der Einführung der OSS sehr aktive Community mit der Zeit einschläft und man sich nach Alternativen für die Weiterentwicklung des Codes umschauen muss. Allerdings lässt sich dieses Risiko durch eine sorgfältige Bewertung der Reife unterschiedlicher OSS-Lösungen (beschrieben in Kapitel *3.2 Qualität und Reifegrad*) minimieren. Eine Herausforderung liegt in der mangelnden vertraglichen Bindung der Community an die eigene Organisation (man kann ein Kollektiv aus freiwilligen Entwicklern kaum in ein Vertragsverhältnis zwingen). Es gilt in diesem Zusammenhang, andere Wege zu finden, die Community zur Entwicklungsarbeit im eigenen Sinne zu motivieren. Auf diese Thematik soll in Kapitel *5.1 Interaktion mit der Community* eingegangen werden. Sollte man den Support durch die Community nicht sicherstellen können, besteht stets die Möglichkeit, diesen von einem externen Dienstleister zu erkaufen.

2.1.5 Einhalten rechtlicher Vorgaben

Für jede Organisation ist die Einhaltung rechtlicher Vorgaben existenziell wichtig. Damit wird die Gewährleistung der Konformität mit Gesetzen und Normen auch bei der Einführung von (Open Source) Software zum entscheidenden Motivationsfaktor.

Im Kontext der Einführung von OSS ergeben sich einige Chancen in diesem Zusammenhang: Möglicherweise kann eine Organisation durch die Nutzung von OSS Vorgaben gerecht werden, die Nutzung offener Formate vorschreiben. So sind beispielsweise in einigen Bundesstaaten der USA alle öffentlichen Stellen gehalten, offene Formate zu verwenden.[7]

[6] http://www.jcp.org/en/home/index
[7] Vgl. Orszag, P. (2009)

Auch ist es denkbar, dass nur eine OSS für ein bestimmtes Anwendungsfeld zertifiziert ist. Ein Beispiel hierfür lässt sich im Bereich der Flugzeug- und Schienenverkehrstechnik finden, für die Linux auf Grund seiner außergewöhnlichen Robustheit empfohlen wird.[8]

Anders herum ist es natürlich denkbar, dass, gerade umgekehrt, keine OSS-Lösung für ein bestimmtes Anwendungsfeld zertifiziert ist bzw. den geforderten Industriestandards entspricht.

Ein weiteres (geringes) Risiko liegt darin, sich durch die modifizierte Weitergabe von OSS mit starkem Copyleft (Software, die diesem Lizenzmodell unterliegenden Code verwendet, unterliegt in der Konsequenz dem selben Lizenzmodell) das Lizenzrecht zu verletzen. Die Angst vor juristischen Auseinandersetzungen kann damit prinzipiell als Herausforderung bei der Einführung von OSS in einer Organisation angesehen werden. Bei reiner Nutzung besteht in dieser Hinsicht jedoch kein Grund zur Sorge. Um dem Leser eine kompetente Beurteilung der juristischen Situation zu ermöglichen, sollen rechtliche, insbesondere lizenzrechtliche, Fragen bei der Einführung von OSS in Kapitel *2.2 Rechtliche Rahmenbedingungen* ausführlich beleuchtet werden.

2.1.6 Erzeugen positiver Außenwirkung

Ein Nebeneffekt der Nutzung von OSS, der als zusätzlicher Motivationsfaktor wirken kann, ist die positive Außenwirkung, die durch Verwendung (und Unterstützung) offener Standards erzeugt wird. Eine Chance einer öffentlichkeitswirksamen Kommunikation dieses Umstandes seitens der Organisation besteht darin, dass sie als Verfechter freiheitlicher Ideale wahrgenommen wird. Wird OSS von ihr nicht nur genutzt, sondern (etwa durch die aktive Teilnahme oder die finanzielle Unterstützung einer Community) sogar gefördert, besteht die Chance, insbesondere junge Menschen auf die Organisation aufmerksam zu machen und diesen ein positives Bild Eigenbild zu vermitteln. Dieser Image-Gewinn kann in unterschiedlichen Dimensionen von Vorteil sein: etwa bei der Mitarbeitersuche, beim Erschließen von Käufergruppen sowie beim Werben um Lieferanten. Auf diesen Aspekt wird in Ansätzen im den Kapitel *5.1 Interaktion mit der Community* eingegangen.

[8] Vgl. Daffara, C. (2009), S. 13

2.1.7 Erzeugen einer positiven Innenwirkung

Ähnlich wichtig wie die Motivation einer positiven Außenwirkung dürfte jene sein, eine positive Innenwirkung zu erzeugen. Die Chance liegt hier darin, dass in der Wahrnehmung der Mitarbeiter die Nutzung (und Unterstützung) offener Formate den Werten der Organisation entspricht. Besonders bei technisch versierteren Nutzern besteht zudem die Chance, dass sie die eigene Organisation als Innovator und Pionier wahrnehmen. Gerade bei weniger Technikaffinen Nutzern besteht jedoch zeitgleich das Risiko, dass sie durch die (ungewohnte) Handhabung der OSS überfordert sind und auf Grund dessen in Resignation verfallen. Hieraus lässt sich als Herausforderung die Angst ableiten, dass die OSS von den Nutzern nicht akzeptiert wird und eine negative Innenwirkung entfaltet. Eine negative Innenwirkung kann im Falle unzureichender Kommunikation der Motivation für die OSS-Einführung potenziell auch durch die Befürchtung hervorgerufen werden, die Organisation wolle an den Arbeitsmitteln sparen. Hierdurch könnten die Produktivität und das Arbeitsklima empfindlich gestört werden. Dieser Problematik und der Frage, wie man als Organisation proaktiv mit ihr umgeht, soll im Kapitel *4.2 Dokumentation und Anwenderschulung* Rechnung getragen werden.

2.1.8 Gewährleistung und Verbesserung der Kommunikation mit Partnern

Eine weitere nennenswerte Motivation für die meisten Organisationen sollte es sein, die Kommunikation mit (strategischen) Partnern zu gewährleisten und, falls möglich, zu verbessern.

Eine Chance bei der Nutzung von OSS liegt darin, dass sich die Software prinzipiell nach Belieben derartig anpassen lässt, dass der an den Partner adressierte Output optimal auf diesen zugeschnitten ist und umgekehrt. Möglicherweise lassen sich die eigenen Softwaresysteme sogar mit denen des Partners verbinden. Gleichzeitig besteht das Risiko, dass ein wichtiger Partner seinerseits ein proprietäres Produkt verwendet, welches nicht mit dem durch die eigene OSS erzeugten Datenformat arbeiten kann. Ein praxisnahes Beispiel für diese Problematik bietet *OpenOffice* bzw. *LibreOffice*, das ein beinahe äquivalentes Produkt zum proprietären *Microsoft Office* darstellt. Die in *OpenOffice* bzw. *LibreOffice* erzeugten Dokumente konnten sich jedoch bis zu Version 2010 von *Microsoft Office* nicht öffnen, geschweige denn

bearbeiten lassen. In der Konsequenz verwenden viele Unternehmen, die intern den Schritt zu offenen Standards gewagt haben, in der Kommunikation mit den Partnern nach wie vor zusätzlich *Microsoft Office*. Hieraus lässt sich als potenzielle Herausforderung die Angst ableiten, dass die Kommunikation mit Partnern unter der Einführung von OSS in der eigenen Organisationen leidet.

Literaturempfehlung:

http://www.phone-soft.at/open-source/10myths_g.html

Liste mit den zehn am weitesten verbreiteten Mythen über Open Source-Produkte und jeweiliger Klarstellung des Sachverhaltes. Sehr geeignet, um unbegründete Vorurteile und Ängste gegenüber OSS auszuräumen.

2.2 Rechtliche Rahmenbedingungen

Mit dem Beleuchten unterschiedlicher Motivationen und Herausforderungen, die mit der Einführung von OSS in Verbindung stehen, ist der erste Schritt getan, eine grundsätzliche Entscheidung für oder gegen die Einführung von OSS in der eigenen Organisation treffen zu können. Ergänzend hierzu soll dem Leser im folgenden Kapitel eine gewisse Sicherheit bezüglich der rechtlichen Rahmenbedingungen im Umgang mit OSS verschafft werden.

In den meisten Fällen wird Open Source Software von Organisationen lediglich genutzt, d.h. nicht modifiziert. Dies gestaltet die Rechtslage sehr überschaubar und hält potentielle Risiken gering. Um jedoch ein Grundverständnis für lizenzrechtliche Themen zu schaffen, sollen in Kapitel *2.2.2 Lizenzarten* die unterschiedlichen Lizenzformen vorgestellt werden. Somit ist ein Unternehmen, das Open Source Software weiterverarbeiten möchte, ebenfalls mit einem Basiswissen ausgestattet. Interessant ist, dass es bisweilen keine „höchstrichterlichen Ent-

scheidungen zu den Rechtsfragen“[9] in Hinblick auf Open Source Software gibt und die wissenschaftliche Fachliteratur die Rechtslage bisher „uneinheitlich beurteilt“[10].

Das folgende Kapitel liefert eine Grundlage an Wissen, die für die reine Nutzung von Open Source Software ausreichend ist. Zieht es ein Unternehmen in Erwägung, Open Source Software zu modifizieren oder weiterzuverbreiten, empfiehlt sich eine tiefere Auseinandersetzung mit der Materie. Hierfür kann ein (ggf. externer) Rechtsexperte zu Rate gezogen werden.

2.2.1 Der Lizenz-Begriff und das deutsche Urheberrecht

Für die rechtlichen Grundlagen von Open Source Software wird das deutsche Urheberrechtsgesetz (kurz UrhG) herangezogen. Der Begriff „Lizenz“ ist im UrhG nicht erwähnt, jedoch werden in der Praxis die Nutzungsrechte an geistigen Werken in Bezug auf Software oftmals als Lizenz bezeichnet. Daher soll im Folgenden der Begriff Lizenz verwendet werden, wenngleich er auch nicht rechtlich einwandfrei ist.

2.2.2 Lizenzarten

2.2.2.1 Strenge Copyleft-Lizenzen

Open Source Software, die unter strengen Copyleft-Lizenzen vertrieben wird, enthält eine Schutzklausel. Deren Ziel ist es, abzusichern, dass „Weiterentwicklungen der Software unter denselben Bedingungen der Lizenz wieder freigegeben werden“.[11] Dadurch soll ausgeschlossen werden, dass OSS in veränderter Form proprietär (also mit geschlossenem Quellcode) vertrieben wird.[12]

Die GNU General Public License

Die am weitesten verbreitete Lizenzart ist die GNU General Public License, kurz GPL. Sie wurde in ihrer zweiten Version 1991 vom Gründer der Free Software Foundation, Richard

[9] Bundesministerium des Inneren (2008), S. 42

[10] ebd.

[11] Jaeger, T./Metzger, A. (2011), S. 5

[12] Vgl. http://www.gnu.org/philosophy/pragmatic.html (Abruf: 29.12.2011)

Stallmann, entworfen und wird als „Grundtypus“[13] der Copyleft-Open Source-Lizenzen gesehen. Ein Copyleft bedeutet, dass die modifizierte Software derselben Lizenz wie der ursprüngliche Quellcode unterliegen muss. Die Hauptbestandteile des Betriebssystems Linux sind beispielsweise durch eine GNU General Public License lizenziert.

Software, die durch eine GPL lizenziert ist, darf nur unter den Bedingungen der GPL weiterverbreitet werden, d.h. für den Nutzer gelten bestimmte Nutzungsrechte für die Weiterverbreitung und Veränderung der Software.

Rechte der Lizenznehmer:

Der Nutzer ist berechtigt, die Software oder Teile der Software zu verändern, zu vervielfältigen und weiterzuverbreiten. Hierbei wird ihm durch den Inhaber der Rechte, z.B. den Entwickler, ein einfaches Nutzungsrecht gemäß §31 Abs.2 UrhG eingeräumt. Seine Rechte darf der Nutzer ausüben, ohne dass beispielsweise Gebühren für eine Lizenz anfallen. Dennoch hat der Nutzer eine Reihe von Pflichten bei der Ausübung seiner Rechte zu beachten.

Pflichten der Lizenznehmer:

Bei den Pflichten des Nutzers wird zunächst unterschieden, ob die Software verändert wurde oder nicht.

Für unveränderte Software gilt, dass

- der vollständige Lizenztext der GPL
- ein Copyrightvermerk
- ein Hinweis auf den Haftungs- und Gewährleistungsausschluss der GPL
- und der Quellcode

beigefügt werden müssen. Des Weiteren dürfen die Pflichten der GPL vom Lizenznehmer nicht erweitert werden.[14]

Für veränderte Software gelten die oben genannten Bedingungen und zusätzlich ein Hinweis darauf, was wann (Datumsangabe) verändert wurde. Alle an der Software vorgenommenen

[13] Jaeger, T./Metzger, A. (2011), S. 25

[14] Vgl. Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V. (o.J.), S. 11

Änderungen werden somit automatisch öffentlich zugänglich. Organisationen sollten sich dessen bewusst sein, um nicht unwillentlich vertrauliche Informationen in den Quellcode einzubauen und diese dann veröffentlichen zu müssen.

Von der zweiten zur dritten Version der GPL:

Zwischen 2005 und 2007 wurde die GPL überarbeitet, sodass 2007 die dritte Version veröffentlicht wurde. In der dritten Version wurden vor allem technische und rechtliche Anpassungen vorgenommen. Ebenso wurde darauf geachtet, dass die Lizenz international angepasst wurde und somit problemlos über Ländergrenzen hinweg verwendet werden kann.[15] In der GNU GPLv3 wurde das Recht des Lizenznehmers erweitert. Es besteht nun die Option, dass der Nutzer, wenn er das Open Source Programm vertreibt, sog. "additional terms", d.h. zusätzliche Lizenzbedingungen beifügen kann.[16]

Die Common Public License[17]

Die Common Public License, kurz CPL, wurde ursprünglich von IBM für deren eigene Open Source Programme entwickelt. Bekannt wurde die CPL hauptsächlich dadurch, dass sie bei der vielgenutzten Entwicklungsumgebung „Eclipse“ Anwendung findet. Für Eclipse gibt es sogar eine eigene Lizenz, die Eclipse Public License (EPL), welche mit der CPL nahezu identisch ist und diese mittlerweile sogar ersetzt hat. In den Rechten der Lizenznehmer unterscheidet sich die EPL von der GPL dadurch, dass sie eine Unterlizenzierung erlaubt. Das bedeutet, ein Lizenznehmer ist befugt, die Lizenz an Dritte weiterzugeben, ohne die Erlaubnis vom ursprünglichen Lizenzinhaber einholen zu müssen. Ein möglicher Nachteil könnte hierbei sein, dass für den Fall, dass der Lizenznehmer selbst an Dritte weiterlizenziert (z.B. das Unternehmen an einen Kunden), eine sogenannte “Rechtekette” entstehen würde. Das bedeutet, der Lizenznehmer (hier: Unternehmen) ist der alleinige Lizenzgeber gegenüber dem Dritten (hier: Kunden); nicht der ursprüngliche Rechteinhaber (hier: Entwickler der OSS). Ist eine Unterlizenzierung nicht gültig (z.B. aufgrund eines ungültigen Vertragsabschlusses), so sind die darauffolgenden Lizenzierungen ebenfalls ungültig.

---

[15] Vgl. Jaeger, T./Metzger, A. (2011), S. 50

[16] ebd., S.54

[17] Der vollständige Lizenztext ist einzusehen unter: http://www.eclipse.org/legal/cpl-v10.html

2.2.2.2 Beschränkte Copyleft-Lizenzen

Die Mozilla Public License

Die Mozilla Public License, kurz MPL, wurde 1998 von dem Unternehmen Netscape für den Browser „Netscape Navigator“ erstellt. Hauptmerkmal der MPL ist es, dass vervielfältigter oder veränderter Quellcode weiterhin MPL-lizenziert bleiben muss. Es schließt jedoch nicht aus, dass dieser Quellcode zusammen mit proprietärem Quellcode genutzt werden kann. So kann der Nutzer eine proprietäre Software als Ganzes entwickeln, welche teilweise MPL-lizenzierten Quellcode enthält.

Rechte des Lizenznehmers:

Die Rechte des Nutzers sind weitgehend mit den Rechten für die Nutzung von Copyleft-lizenzierter Software identisch.

Pflichten des Lizenznehmers:

Bei den Pflichten des Nutzers ist ein merklicher Unterschied zu den strengen Copyleft-Lizenzen zu erkennen. Bei einer beschränkten Copyleft-Lizenz wie der MPL hat der Nutzer weniger Pflichten, sodass er lediglich

- die Lizenz weitergeben
- den Copyrightvermerk beibehalten
- und einen Hinweis auf den Haftungsausschluss der Lizenz hinzufügen muss.[18]

Zusammenfassend lässt sich sagen, dass sie MPL in der Praxis gerne als Kompromiss zwischen der GPL und der Copyleft-freien BSD (vgl. Kapitel *2.2.2.3 Lizenzen ohne Copyleft*) verwendet wird, da sie ein „abgeschwächtes“ Copyleft enthält.

Die GNU Lesser General Public License[19]

Die GNU Lesser General Public License (LGPL) orientiert sich weitgehend an der GPL. Sie unterscheidet sich von letzterer im Wesentlichen durch ihre abgewandelten Richtlinien in Bezug auf Softwarebibliotheken.

---

[18] Vgl. Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V. (o.J.), S. 11
[19] Der vollständige Lizenztext ist einzusehen unter: http://www.gnu.org/licenses/lgpl.html

In der Praxis bedeutet dies, dass eine Software, die Teile von LGPL-lizenzierter Software verwendet, nicht zwingend LGPL-lizenziert werden muss, sondern ihre eigene Lizenzart beibehalten kann. Dies ist vor allem für Softwarebibliotheken von Vorteil, die so für proprietäre Software verwendet werden können. Zu unterscheiden sind jedoch die Verwendung und die feste Einbettung einer LGPL-lizenzierten Software in eine andersartig lizenzierte. Soll eine Einbettung stattfinden, so muss die gesamte Software LGPL-lizenziert werden oder durch eine mit der LGPL kompatible Lizenz ausgestattet sein.

Ein möglicher Vorteil der LGPL ist es, dass veränderte Software problemlos eine GPL annehmen kann, falls eine strenge Copyleft-Lizenzierung für die veränderten Softwarebestandteile gewünscht wird.

2.2.2.3 Lizenzen ohne Copyleft

Die Berkeley Software Distribution-Lizenz

Ein Beispiel für eine Copyleft-freie Lizenz ist die Berkeley Software Distribution-Lizenz, kurz BSD-Lizenz, die in den 1970er Jahren an der Universität Berkeley für die Lizenzierung einer Unix-Variante entwickelt wurde.[20]

Lizenzen ohne Copyleft setzen den Schwerpunkt auf eine Maximierung der Nutzungsrechte bei gleichzeitiger Minimierung der damit verbundenen Nutzerpflichten. Aufgrund der geringeren Pflichten des Lizenznehmers kommt es hier zu weniger rechtlichen Uneinigkeiten.

Ein besonderer Vorteil der copyleftfreien Lizenzen ist es zudem, dass vorgenommene Änderungen an der Software auch unter eigens erstellten Lizenzbedingung weiterverbreitet werden dürfen. Dadurch wird eine Verwendung mit proprietärer Software möglich. Es ist sogar erlaubt, die veränderte Software unter proprietären Lizenzen zu vermarkten, ohne dabei den ursprünglichen Quellcode anzugeben.

Bei der BSD müssen Urhebervermerk, Lizenzbestimmungen und Gewährleistungsausschluss (vgl. Copyleft-Lizenzen) mitverbreitet werden. Wenn eine Originalversion verwendet wird, muss erwähnt werden, dass die Universität Berkeley die BSD entwickelt hat, wird sie jedoch modifiziert, fällt selbst dieser Vermerk weg.

[20] Vgl. Jaeger, T./Metzger, A. (2011), S. 84

Allgemein lässt sich sagen, dass wenn die BSD bearbeitet wurde, keine Pflichten entstehen. Demzufolge ist auch für den Fall eines Verstoßes gegen Pflichten, die es bei Modifikation quasi nicht gibt, in der BSD keine Information hinterlegt.

2.2.2.4 Sonstige Lizenzformen

Die oben genannten Lizenzarten werden durch die sogenannten „Artistic Licences“ ergänzt. Bei diesen ist es dem Nutzer freigestellt, beim Weitergeben der veränderten Software aus bereits vorgegebenen Lizenzbedingungen zu wählen. Ein Beispiel einer Artistic-lizenzierten Software ist die Programmiersprache „Perl“.

Daneben gibt es noch über 40 weitere Lizenzarten, welche jedoch aufgrund ihrer geringeren Popularität hier nicht namentlich erwähnt werden sollen.

2.2.3 Urheberrechtliche und vertragsrechtliche Fragen

Gemäß §2 Abs. 2 UrhG werden durch das Urheberrecht „persönliche geistige Schöpfungen“, so auch Open Source Software, geschützt. Der Schutzumfang umfasst hierbei sowohl die persönlichen Beziehungen des Urhebers zu seiner Schöpfung („Urheberpersönlichkeitsrecht“) als auch die Nutzung der Schöpfung („Verwertungsrecht“). Im Allgemeinen darf eine Schöpfung nur durch den Urheber verwendet werden. Letzterer kann jedoch zusätzlichen Personen sogenannte „Nutzungsrechte“ einräumen. Diese Rechte werden in einem „Lizenzvertrag“ niedergelegt. Er enthält somit sowohl urheberrechtliche als auch vertragsrechtliche Aspekte. Es ist zu beachten, dass bei einem „Softwareüberlassungsvertrag“ nicht automatisch eine Software überlassen wird, sondern vielmehr die Einräumung von Rechten durch den Vertrag geregelt wird.[21] Somit ist es wichtig, bei der Überlassung von Open Source Software zwischen der Software selbst, „Bits und Bytes“ und den Nutzungsrechten an der Software, der „Lizenz“ zu unterscheiden.[22]

Wird Open Source Software lediglich zum Ablaufenlassen, Erstellen einer Sicherungskopie und zum Berichtigen von Fehlern verwendet, so wird kein Lizenzvertrag benötigt.

---

[21] Vgl. Koglin, O./Metzger, A. (2004), S. 6

[22] Vgl. Bundesministerium des Inneren (2008), S. 43

Soll hingegen eine Vervielfältigung, Veränderung, Verbreitung oder/und öffentliches Zugänglichmachen betrieben werden, ist der Abschluss eines Lizenzvertrages notwendig. Hierbei ist auch ein "stiller" Lizenzvertrag ausreichend. Dabei handelt es sich um einen Vertrag, der durch die Annahme der Lizenzbedingungen durch den Nutzer zu Stande kommt. Ein schriftlicher Vertrag muss an dieser Stelle nicht vorliegen.

2.2.4 Nutzung von Open Source Software ohne Modifikationsabsichten

Unabhängig von der Lizenzart, ist dem Nutzer die reine Nutzung der Open Source Software in jedem Fall gestattet. Dies steht ausdrücklich in der zweiten Version der GPL "Activities other than copying, distribution and modification are not covered by this license; they are outside its scope. The act of running the program is not restricted [...]."[23] sowie in der dritten Version "You are not required to accept this license in order to receive or run a copy of the program."[24]

Nach dem Urheberrechtsgesetz ist es ebenso gestattet, eine Sicherungskopie (§ 69d Abs. 2 UrhG) zu erstellen und Fehler an der Software zu korrigieren (§ 69d Abs. 1 UrhG). Werden von einem Programm eine Vielzahl von Sicherheitskopien erstellt und weitergegeben, oder findet eine Veränderung, die über eine Fehlerbehebung hinausgeht, statt, so ist der Abschluss eines (stillen) Lizenzvertrags erforderlich. Andernfalls nicht.[25]

Zusammenfassend lässt sich demnach festhalten, dass für das reine Ausführen von Open Source Software kein Abschluss eines Lizenzvertrages notwendig ist.

2.2.5 Checkliste

Wie soll die Open Source Software verwendet werden?

1. Soll die Software laufen gelassen werden?

Dies dürfte in der Regel bei jeder Software der Fall sein, so auch im betrachteten Beispielunternehmen. Hier ist nichts weiter zu beachten, es wird kein Lizenzvertrag eingegangen.

---

[23] Vgl. Ziffer 0 Absatz 2, GPL Version 2

[24] Vgl. Ziffer 9, GPL Version 3

[25] Vgl. Bundesministerium des Inneren (2008), S. 45

2. Soll eine Sicherungskopie erstellt werden?

Dies ist i.d.R. empfehlenswert, auch im Beispielunternehmen sollten Sicherungskopien erstellt werden. Hier ist nichts weiter zu beachten, es wird kein Lizenzvertrag eingegangen

3. Sollen Fehler berichtigt werden?

Sollten in der Open Source Software Fehler auftreten, welche für den ordnungsgemäßen Ablauf der Software von Bedeutung sind, dürfen und sollten sie behoben werden.
Hier ist nichts weiter zu beachten, es wird kein Lizenzvertrag eingegangen.

4. Soll die Open Source Software vervielfältigt werden?

Das Beispielunternehmen plant, mit der Open Source Software eine Vielzahl von Arbeitsplätzen auszustatten. Hier muss nun das deutsche Urheberrecht angewandt werden, es ist von einer Vervielfältigung auszugehen. Eine Open Source Software-Lizenz ist abzuschließen. Dies kann auch in Form eines "stillen" Vertrages, d.h. keiner schriftlichen Ausfertigung, sondern lediglich der Zustimmung des Nutzers, stattfinden.
Zu beachten:

- Ausfindig machen, unter welcher Lizenz die Open Source Software läuft.
- Sich über die Rechte und Pflichten des Nutzers bewusst werden, insbesondere wenn es sich um eine strenge oder beschränkte Copyleft-Lizenz handelt.
- Bei der Verteilung prüfen, ob die Nutzerpflichten eingehalten wurden.

5. Soll die Open Source Software verändert werden?

Im Beispielunternehmen steht dies nicht zur Debatte.

6. Soll die Open Source Software (außerhalb der eigenen Organisation) vertrieben werden?

Im Beispielunternehmen steht dies nicht zur Debatte.

7. Soll die Open Source Software öffentlich zugänglich gemacht werden?

Im Beispielunternehmen steht dies nicht zur Debatte.

Literaturempfehlung:

Jaeger/Metzger (2011): Open Source Software: Rechtliche Rahmenbedingungen der Freien Software, 3. Auflage, München: Beck Juristischer Verlag

Sehr umfassendes Werk zum Thema Rechtliches von Open Source, verschiedene Lizenzarten sowie Einführung in das Vertrags- u. Urheberrecht.

Bundesministerium des Inneren (2008): Migrationsleitfaden: Leitfaden für die Migration von Software, Version 3.0, 1. Auflage, Berlin

Ab Seite 41 werden rechtliche Aspekte von Softwaremigrationen vorgestellt.

Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V.

2.3 Was die folgenden Kapitel des Ratgebers leisten

Im Verlauf des Kapitels *2. Orientierungsphase* sollte der Leser ein solides Grundverständnis zum Thema OSS und ein Gespür für relevante Überlegungen bezüglich deren Verwendung entwickelt haben. Die folgenden Kapitel sollen ihm helfen, sein Wissen zu vertiefen und seinen Blick für eine Vielzahl von entscheidungsrelevanten Dimensionen bei der Evaluation, Transformation und beim späteren Betrieb von OSS zu schulen.

Durch das Lesen des restlichen Ratgebers wird der Leser die Kompetenz erlangen, die hier als Schablone dargebotene zusammenfassende Matrix (auf seinen Anwendungsfall bezogen) zielführend zu befüllen und hieraus eine fundierte Entscheidung für oder gegen die Einführung einer OSS zu fällen. Die Matrix enthält keine mathematischen Formeln und wird das Ergebnis der Evaluation nicht auf eine Zahl herunterbrechen. Dies ist der Tatsache geschuldet, dass sich die allermeisten entscheidungsrelevanten Dimensionen nicht auf einen numerischen Wert herunterbrechen lassen. Die Matrix kann als Checkliste für und Übersicht über eine Vielzahl sinnvoller Weise zu berücksichtigender Fragestellungen verstanden werden. Sie der eigenen Organisation und konkreten Software entsprechend zu modifizieren und sinnvoll zu

befüllen, obliegt dem Leser. Die Gewichtungen von “0 - ohne Relevanz” bis “10 - von höchster Relevanz” sind nach eigenem Ermessen vorzunehmen.

Am Ende des Ratgebers wird (quasi als Synthese aller vorangegangenen Kapitel und Checklisten) die hier vorgestellte Matrix für das bereits vorgestellte Referenzunternehmen ausgefüllt dargeboten werden.

Entscheidungskriterium	**Wert/Antwort**	**Relevanz**	**K.O.-Kriterium?**
Ideologische Motivation (z.B. Freiheitsgedanke)			
Einhaltung rechtlicher Rahmenbedingungen			
Einschätzung Außenwirkung			
Einschätzung Innenwirkung			
Kommunikation mit Partnern			
Ist die Software relevant für Schnittstellen mit Partnern?			
Wie hoch ist die Kompatibilität der Datenformate mit den Industriestandards?			
Wie gut sind die Möglichkeiten, Datenformate zu ändern?			
Gesamteinschätzung			
Rechtliche Rahmenbedingungen			
Wird die Software modifiziert (oder weitervertrieben) und ergeben sich daraus rechtliche Konsequenzen?			
Stellt uns die Pflicht zur Freigabe von Quellcodeänderungen vor Probleme (Wahrung von Betriebsgeheimnissen)?			
Gesamteinschätzung			

Gesamteinschätzung unmittelbare finanzielle Auswirkung			
Anforderungserfüllung			
Wie gut werden die funktionalen Anforderungen erfüllt?			
Wie gut werden die nicht-funktionalen Anforderungen erfüllt?			
Gesamteinschätzung			
Qualität und Reifegrad von Software und Community			
Wie ist die Softwarequalität einzuschätzen?			
Welchen Reifegrad hat die Community?			
Gibt es professionelle Supportangebote?			
Gesamteinschätzung			
Migration			
Ist eine Migration technisch möglich?			
Wird externer Sachverstand benötigt um die Migration durchzuführen?			
Gesamteinschätzung			

Tabelle 1: Entscheidungsmatrix

3 Evaluationsphase

Am Ende der Orientierungsphase sollte man durch Analyse der eigenen Motivationsstruktur und Klärung der rechtlichen Rahmenbedingungen evaluiert haben, ob eine Open Source-Lösung prinzipiell für die Einführung in der Organisation in Betracht kommt. In der Evaluationsphase gilt es in einem nächsten Schritt, konkrete Software-Lösungen mit den Anforderungen der Organisation abzugleichen und sie auf ihre Kostenstruktur hin zu untersuchen und in letzter Konsequenz eine Entscheidung zu fällen.

3.1 Auswahl der passenden Software

Ein essenzieller Schritt bei der Einführung eines neuen Programms in einer Organisation ist die Auswahl der Software. Um das passende Programm zu finden, steht zunächst eine Frage im Mittelpunkt: Welche Aufgaben möchte die Organisation mit der Software in welchem Umfang lösen? Man spricht hier davon, die Anforderungen an die Software zu sammeln und zu analysieren. Im Verlauf dieses Kapitels wird daher erläutert, worauf bei der Anforderungsanalyse geachtet werden muss, welche Methodik genutzt werden kann, um die Produktauswahl einzugrenzen und welche Möglichkeiten sich durch die Wahl von OSS (im Gegensatz zu CSS) zusätzlich bieten.

Um ein passendes OSS-Produkt auszuwählen ist es, analog zu kommerziellen Produkten, nötig, eine genaue Übersicht darüber zu erstellen, welche funktionalen und nicht-funktionalen Anforderungen das Produkt erfüllen muss. Funktionale Anforderungen beschreiben dabei die benötigten Fähigkeiten, um eine bestimmte Aufgabe zu lösen (z.B. das Erstellen eines Textdokumentes oder das Speichern von relationalen Daten). Nicht-funktionale Anforderungen sind solche, die die technische Ausgestaltung dieser Fähigkeiten betreffen (u.a. Geschwindigkeit, Zuverlässigkeit). Eine solche nicht-funktionale Anforderung kann beispielsweise das stabile Zustellen von Mails auf einem Server darstellen, der 50 Mails in der Sekunde verarbeiten muss.

3.1.1 Funktionale Anforderungen

Das vorhandene Wissen über die benötigten Funktionen wird sich stark unterscheiden, je nachdem, ob das OSS-Produkt eine komplette Neueinführung ist, oder lediglich eine bestehende Softwarelösung ersetzt. Um dem Rechnung zu tragen, sollte je nach Art der Software eine unterschiedliche Vorgehensweise angewendet werden. Je umfangreicher und neuer die Aufgabe, desto eher sollte die Vorgehensweise use-case-orientiert (1) sein um alle Anforderungen zu erfassen. Bei simplen Aufgaben bzw. Ablösungen kann die Vorgehensweise an eigenen oder auf dem Markt vorhanden Lösungen orientiert (2) sein, insofern dies möglich ist.

1. Bei der use-case-orientierten Vorgehensweise werden von allen Benutzerrollen in Interviewform Informationen darüber gesammelt, welche Aufgaben sie konkret mit der Software lösen möchten. Unter einem use-case oder auch Anwendungsfall versteht man dabei die Menge an Aktionen, die der Nutzer mit der Software ausführt, um ein bestimmtes Ergebnis zu erzielen. Die Benutzer vergeben dabei bereits Prioritäten, um die Unterscheidung zwischen zwingend benötigten und optionalen Funktionen zu ermöglichen. Essentiell ist dabei, alle Benutzerrollen abzudecken, da je nach Software starke Unterschiede zwischen den Rollen existieren können (z.B. ist es bei einer Groupwarelösung für Nutzer wichtig, einen Client zu haben, der Mail,- Kalender-, und Aufgabenverwaltung unterstützt, während die Administratoren zusätzlich einen Server benötigen, der eine Begrenzung des Serverspeichers pro Benutzer erlaubt und das Senden von out-of-office-Nachrichten). Diese Informationen lassen sich u.a. in UML[26] use-case-Diagrammen visualisieren, um eine schnellere Übersicht zu ermöglichen aber auch, um Missverständnisse zu vermeiden. Aus der Sammlung der use-cases können nun Funktionale Anforderungen abgeleitet werden. Dabei ist die Priorität heranzuziehen, die einzelnen Nutzerrollen beigemessen wird, um K.O.-Kriterien zu definieren und die weiteren Funktionen zu gewichten.

2. Alternativ zur zeitintensiven use-case-orientierten Vorgehensweise ist es auch möglich, sich an bereits bestehenden Lösungen zu orientieren. Dabei ist zu analysieren, welche Funktionen die aktuelle Lösung bereitstellt, um dann zwei Fragen zu beantworten.

[26] http://www.uml.org/ - für eine deutschsprachige Übersicht bietet sich http://www.umlsig.de/ an

a. Welche vorhandenen Funktionen werden benötigt? Welche nicht? Ein zu großer Funktionsumfang kann dazu führen, dass Nutzer und Administratoren unnötigen Aufwand durch Konfiguration oder Suche nach den eigentlich nötigen Funktionen haben. Außerdem vergrößert potentiell jede Funktion die Gefahr eines Fehlers in der Software.

b. Welche zusätzlichen Funktionen werden benötigt? Hier gilt es, kreativ zu sein: Welche Schritte müssen noch manuell durchgeführt werden? Wo existieren Medienbrüche oder vermeintlich unnötige Wechsel zwischen Programmen?

Diese Vorgehensweise kann vor allem bei einer Ablösung ohne große Funktionsänderung schneller durchgeführt werden als eine use-case-orientierte, die bei Null anfängt. Allerdings besteht die Gefahr, dass bei nicht ausreichend kritischer Betrachtung notwendige Funktionen vergessen werden oder nicht benötigte die Auswahl der Software erschweren bzw. verzerren.[27] Anhand der geforderten Funktionen kann jetzt eine Liste mit Produkten erstellt werden, die den Funktionsbereich abdecken bzw. vermeintlich abdecken. Hierbei ist zwischen OSS und kommerzieller Software keine Unterscheidung notwendig, eine (vor allem aufgrund der regelmäßigen Aktualisierung) geeignete Quelle ist hier eine Übersicht nach Softwaregruppen auf Wikipedia[28], zusätzlich können die im Literaturtipp genannten Internetseiten durchsucht werden.

3.1.2 Nicht funktionale Anforderungen

Nicht-funktionale Anforderungen sind oft schwerer zu fassen als funktionale, da sie sich auf die Eigenschaften einer Software beziehen, quasi auf die Details unter der Haube. Nichtsdestotrotz sind sie genauso wichtig.

Während die funktionalen Anforderungen vor allem von den gegenwärtigen tatsächlichen Bedürfnissen abhängen, können bei nicht-funktionalen auch Zukunftspläne oder historische Komponenten sowie schwer messbare subjektive Kriterien eine große Rolle spielen.

[27] Eine detaillierte Beschreibung dieser Vorgehensweise ist unter http://www.stickyminds.com/pop_print.asp?ObjectId=3430&ObjectType=ART vorhanden.

[28] https://en.wikipedia.org/wiki/Free_alternatives_to_proprietary_software

Volere kategorisiert nicht-funktionale Anforderungen wie folgt[29] :

- Benutzbarkeit - Ist die Qualität der Software, vom Nutzer gebraucht zu werden und an ihn angepasst zu werden ausreichend? Darunter fällt z.B. die Übersetzung in die jeweilige Sprache oder auch, wie einfach ein Programm zu erlernen und nutzen ist.
- Performance - Welche Kapazität, Geschwindigkeit der Anfragenbeantwortung oder welcher Zuverlässigkeitsgrad werden von der Software erwartet?
- Ausführungsumgebung - Auf welcher Hardware läuft die Software? Welche Schnittstellen gibt es zu anderen Systemen?
- Wartung und Support - Wie viel Zeit darf die Wartung benötigen? Welcher Supportaufwand ist tolerierbar?
- Sicherheit - Schützt die Software die genutzten Daten ausreichend vor unberechtigtem Zugriff, vor Beschädigung und werden dabei rechtliche Vorgaben eingehalten?
- Kulturell und Politisch - Wie müssen Zahlen formatiert werden? Wie die Uhrzeit dargestellt? Müssen Führungskräften mehr Funktionen zur Verfügung stehen als ihren Mitarbeitern?
- Rechtlich - Wie lang muss die Software Daten archivieren? Welche Daten dürfen nicht erfasst werden?

Besonders kritisch sind hier Anforderungen bezüglich der Schnittstellenkompatibilität. Dabei darf nicht nur auf Schnittstellen zu Software in der eigenen Organisation geachtet werden, sondern es müssen auch alle Schnittstellen und die hieraus resultierende Notwendigkeit von Kompatibilität mit Partnern und Kunden miteinbezogen werden. Während Schnittstellen von einer OSS zur anderen notfalls auf Anfrage oder Bezahlung entwickelt werden können, bestehen bei Schnittstellen von und zu kommerzieller Software folgende Probleme:

- fehlende Dokumentation der Schnittstelle seitens des Herstellers
- oder eines der vier in Kapitel *3.1.4 Anpassung der Software an die Anforderung* genannten Probleme

[29] http://www.volere.co.uk/template.htm, für weiterführende Informationen: http://www11.informatik.uni-erlangen.de/Lehre/WS0809/PR-SWE/Material/volere-template.pdf

3.1.3 Methodik zur Auswahl passender Software

Nach der Sammlung von funktionalen und nicht-funktionalen Anforderungen und entsprechender potentieller OSS Lösungen bietet es sich an, für jeden der beiden Anforderungstypen eine Matrix zur Unterstützung des Auswahlprozesses zu erstellen. Die Auflistung der Anforderungen und Gewichtungen in einer Tabelle ermöglicht eine Übersicht darüber, inwieweit untersuchte Programme die Anforderungen erfüllen. Die Skalen für die Gewichtung und die jeweilige Erfüllung sehen dabei wie folgt aus:

- Gewichtung: numerisch, sei es eine prozentuale Gewichtung oder eine Notenskala (z.B. 1-10); zusätzlich die K.O-Option.
- Grad der Erfüllung: numerisch, mit einer geraden Wertemenge (z.B. von 1-4, da beim Ausfüllen im Zweifelsfall die Versuchung groß ist, die mittige Option auszuwählen), wobei ein K.O. Kriterium nur zwei mögliche Erfüllungsgrade kennt: "Ja" und "Nein".

Um am Ende zu sehen, welches Produkt am besten geeignet ist, wird nun die Gewichtung mit dem Erfüllungsgrad multipliziert und die Ergebnisse für jedes Produkt summiert. Produkte die ein oder mehrere K.O.-Kriterien nicht erfüllen, fallen dabei aus der weiteren Betrachtung heraus. Die Beispieltabelle (siehe Tabelle 1) für die funktionalen Anforderungen zeigt diesen Fall[30]: Produkt A wird - obwohl es mehr Punkte als Produkt B hat - aus der weiteren Auswahl herausfallen (beachte Kapitel *3.1.4 Anpassung der Software an die Anforderungen*).

---

[30] Die Beispieltabelle für die nicht-funktionalen Anforderungen ist identisch aufgebaut.

Anforderung an die Groupwarelösung	Gewichtung (1-10)	Prod. A	Prod. A gewichtet	Prod. B	Prod. B gewichtet
Client - Verwaltung von Aufgaben	K.O.	Nein	-	Ja	-
Client - Synchronisierung des Adressbuchs mit Mobiltelefonen	3	3	9	3	9
Server - Verwaltung von Speicherplatzquota	9	4	36	3	27
Summe/K.O. erfüllt?	**-**	**Nein**	**45**	**Ja**	**36**

Tabelle 2: Anforderungserfüllung mit Gewichtung

3.1.4 Anpassung der Software an die Anforderungen

Sollte kein Produkt alle funktionalen und nicht-funktionalen Anforderungen abdecken, ist man bei kommerzieller Software darauf angewiesen, dass der Hersteller (durch Entlohnung) überzeugt werden kann, die benötigten Anpassungen vorzunehmen. Mögliche Probleme könnten hierbei sein, dass der Hersteller:

1. einen zu hohen Preis fordert,
2. nicht das nötige Knowhow hat,
3. keine Kapazitäten hat,
4. sich aus strategischen Gründen weigert (z.B. eine Exportfunktion zu implementieren, die Wechselbarrieren abbauen würde)

Während die genannten Probleme prinzipiell auch bei OSS bestehen können, ist dies hier jedoch deutlich unwahrscheinlicher. Die Community hinter einem OSS-Projekt ist bezüglich der Zusammensetzung der Entwickler oft größer und diversifizierter aufgestellt als ein kommerzielles Softwareunternehmen. Zusätzlich kann bei Bedarf jeder hierzu fähige Entwickler den Code einsehen und anpassen. Dadurch ist

1. der Preis für Anpassungen geringer, da mehr Konkurrenz herrscht,

2. das vorhanden Knowhow größer, wobei mangelndes Wissen anhand der offenen Quellen im Bedarfsfall aufgebaut werden kann (was allerdings zu mehr Zeitaufwand und/oder Kosten führt)
3. der Pool an Entwicklern, aus dem Auftraggeber schöpfen können deutlich größer, wodurch die Chance erhöht wird freie Kapazitäten zu finden,
4. stets die Möglichkeit gegeben, dem Projekt nicht zugehörige Entwickler, (die der Projektstrategie nicht verschrieben sind) für Änderungen zu bezahlen. Hinzu kommt, dass der Gedanke der Offenheit, der in der Open-Source-Bewegung herrscht, normaler Weise dafür sorgt, Wechselbarrieren abzubauen.

Zusätzlich besteht die Möglichkeit, die gewünschten Änderungen kostenlos zu erhalten, wenn man es schafft die Community hiervon zu überzeugen (die zeitliche Planung ist hier allerdings schwieriger als bei einer Änderung gegen Bezahlung). Sollte die Änderung von der Organisation selbst oder in ihrem Auftrag durchgeführt werden, ist dringend darauf zu achten welche Rechte und vor allem Pflichten aus der jeweiligen Open Source-Lizenz entstehen (vgl. Kapitel *2.2 Rechtliche Rahmenbedingungen*).

3.1.5 Auswahltests

Um zu Überprüfen, ob tatsächlich alle Anforderungen fehlerfrei umgesetzt sind sollte stets ein Test der OSS durchgeführt werden. Hierbei gilt es, zu unterscheiden zwischen Tests, die noch Teil des Auswahlprozesses sind, und Anwendertests, die für eine konkrete Software durchgeführt werden.

Tests als Teil der Auswahl empfehlen sich erst dann, wenn die nur noch eine geringe Zahl an Programmen zur Auswahl steht. Besonders sinnvoll ist ein Auswahltest, wenn die Ergebnisse der einzelnen OSS-Projekte in der Matrix nah aneinander liegen. Bei diesem Tests sind noch keine Nutzer beteiligt. Auswahltests können Faktoren hervorbringen, die in der Matrix nicht numerisch erfasst waren (z.B. die Schwierigkeit der Konfiguration). Am Ende des Auswahltests muss die Entscheidung getroffen werden, welches das eine Produkt ist, dass die Anforderungen am besten abdeckt und demnach in den Anwendertest kommt.

3.1.6 Anwendertests

Unter Anwendertest ist ein „Test der Gesamtanwendung zur ganzheitlichen Überprüfung der Erfüllung der [...] Anforderungen im Rahmen einer simulierten Zielorganisation“[31] zu verstehen. Anforderungen sind für die Open Source-Einführung in erster Linie die Akzeptanz der neuen Softwarelösungen unter den Mitarbeitern. Ob ein Anwendertest sinnvoll ist, hängt von dem Umfang der OSS ab, die in einem Unternehmen eingeführt werden sollen. Da ein Anwendertest (nicht zuletzt dadurch, dass er die Zeit von Testpersonen und Verantwortlichen in Anspruch nimmt) Kosten verursacht, sollte im Vorfeld untersucht werden, welche Bedeutung der Akzeptanz der Mitarbeiter beigemessen wird. Handelt es sich bei der einzuführenden Software um eine häufig genutzte Anwendung, mit der eine Großteil der Mitarbeiter in regelmäßigem Kontakt steht, ist ein Anwendertest zu empfehlen.

Bei der Durchführung eines Anwendertest können folgende Handlungsschritte abgearbeitet werden[32]:

1. Vorbereitung
2. Testgruppe / Testmethode ausfindig machen
3. Planung und Ausführung
4. Dokumentation der Testergebnisse & Auswertung
5. Zusammenfassung

In ersten Schritt sollte definiert werden, was getestet werden soll. Dies kann eine komplette Anwendung sein oder nur einzelne Funktionalitäten einer Anwendung. Die Testgruppe sollte die spätere Anwendergruppe repräsentieren. Eine Auswertung der Testergebnisse kann Hinweise darauf geben, wie die OSS von den Anwendern angenommen wird. Fällt ein Anwendertest negativ aus, sollte überdacht werden, ob eine Einführung Sinn ergibt und wenn ja, wie diese Skepsis überwunden werden kann. Eine denkbare Möglichkeit besteht darin, ein entsprechendes Schulungsangebot und Lernmaterial für die Anwender bereitzustellen. Auf diesen Ansatz wird in Kapitel *4.2 Anwenderschulung und Dokumentation* näher eingegangen.

[31] Sladic, D. (2005), S. 5

[32] Vgl. http://www.causause.de/leistungen/anwendertest.html

3.1.7 Checkliste

1. Sind die funktionalen Anforderungen von allen relevanten Rollen gesammelt?

(Rollen sind u.a. die durchschnittlichen Nutzer, Führungskräfte, Administratoren.)

Ja: Alle funktionalen Anforderungen sind gesammelt

Für den Client:

- Mailfunktion
- Kalenderfunktion
- Aufgabenverwaltung
- Adressbuch
- lokale Archivierungsmöglichkeit

Und für den Server:

- Mail, Kalender, Aufgabenverwaltung
- Erlauben von Speicherplatzbeschränkungen

Nein:

Die Sammlung der Anforderungen ist nicht vollständig. In diesem Fall sollte dringend beachtet werden, dass fehlende Funktionen später zu großen Problemen und Unzufriedenheit von Seiten der Mitarbeiter führen können.

2. Sind die nicht funktionalen Anforderungen aller Rollen gesammelt?

Ja: Alle nicht funktionalen Anforderungen sind gesammelt

Client

- Auch auf schwachen älteren Rechnern lauffähig
- Verschlüsselung aller Daten und Datentransfers
- Auf Windows und Linux lauffähig Server
- Wartung/Administration ohne extra Client
- Skalierbar auf mehrere Server ohne Performanceeinbußen

Nein: siehe oben.

3. Wurde eine Matrix mit gewichteten Anforderungen für jedes potentielle Produkt ausgefüllt? Wurde das Programm ausgewählt, das die Anforderungen am besten abdeckt?

Ja, die Tabelle wurde analog Tabelle 1 ausgefüllt. Zwei Produkte hatten jedoch fast die gleiche Punktzahl.

4. Wurden mit allen in Frage kommenden Produkte ein Auswahltest durchgeführt?

Installation von 2 Produkten wurde durchgeführt, dabei wurde festgestellt, dass der Client der einen Lösung zwar lauffähig aber unter Windows wenig performant und instabil ist.

5. Haben die Nutzer die Software im Test akzeptiert?

Die Nutzer fanden alle benötigten Funktionen vor. Für die Nutzung brauchten sie jedoch etwas mehr Anleitung als erwartet, daher wird eine planmäßige Schulung der Nutzer nötig sein, während ursprünglich gehofft wurde, dass das Bereitstellen einer simplen Einführungspräsentation ausreichen würde.

Literaturempfehlung:

OSS-Projekthosting-Seiten:

http://www.sourceforge.net/

http://savannah.gnu.org/

https://www.gna.org/

http://alioth.debian.org/

http://www.berlios.de/

http://www.codehaus.org/

http://code.google.com

OSS-Softwarekataloge:

http://www.freshmeat.net/

http://www.eosdirectory.com/

http://sourcewell.berlios.de/

Liste mit Gegenüberstellungen von CSS zu OSS:

http://www.linuxrsp.ru/win-lin-soft/table-eng.html

http://en.wikipedia.org/wiki/Free_alternatives_to_proprietary_software

http://www.osalt.com

3.2 Qualität und Reifegrad

Bei der Auswahl von Software spielen neben den technischen und Kostenaspekten auch die Qualität der Software sowie die um das Projekt existierende Community eine Rolle.

3.2.1 Bewertung der Community

Da beim Einsatz von OSS nicht zwingend ein Hersteller als Vertragspartner vorhanden ist, der für etwaige Qualitäts- oder Sicherheitsmängel gerade zu stehen hat, spielt die Community eine besonders wichtige Rolle. Die Community sollte im Idealfall drei distinkte Leistungen erbringen:

1. (Weiter-) Entwicklung
2. Fehlerbehebung
3. Support

Eine Community lässt sich im Gegensatz zu einem Unternehmen nicht vertraglich binden, da man es mit einer Vielzahl von Entwicklern zu tun hat, die für unterschiedlichste Unternehmen oder in ihrer Freizeit an einem Programm mitarbeiten (z.B. liegt die Zahl der Linux Entwickler bei über 1000 verteilt auf über 200 Firmen[33]). Es liegt daher an der Organisation

[33] http://www.desktoplinux.com/news/NS7450801259.html

eine Abschätzung zu erstellen, wie stabil die aktuelle Community ist und in welche Richtung sie sich bewegt. Neben einer tiefgehenden Analyse über die Motivationen von Entwicklern und die daraus abgeleitete Stabilität der Community[34], kann man auf Kennzahlen zurückgreifen, die sich aus den öffentlichen Plattformen eines Projektes erstellen lassen. Die von mittlerweile leider inaktiven OSS-Bewertungsprojekten wie FLOSSMetrics[35] oder SQO-OSS[36] genutzten Quellen sind die Quellcodeverwaltung, die Mailingsliste und der Bugtracker. Aus diesen lassen sich verschiedene absolute Zahlen und Verhältnisse herauslesen, die entweder Aussagen über die Stabilität der Community an sich oder deren Entwicklungsverhalten erlauben. Interessant ist beispielsweise die Zahl der aktiven Entwickler, die Entwicklung der Anzahl der Hauptentwickler (jene, die den Großteil des Codes beisteuern) oder die durchschnittliche Dauer, die Entwickler in der Community aktiv sind. Darüber hinaus können u.a. folgende Kennzahlen wertvoll sein:

- der Anteil der Entwickler, die an älteren Versionen arbeiten
- die Anteile des Quellcodes, die jeweils von nur einem Entwickler angefasst wurde und für die das Ausscheiden dieses Entwicklers zu einem starken Knowhow-Verlust führen würde

Die Erfassung dieser Kennzahlen ist jedoch mit einigem Aufwand verbunden, der oft nicht gerechtfertigt ist, sei es weil die Kosten zu hoch sind, nicht das nötige Wissen in der Organisation bereitsteht, oder schlicht schnellere und einfachere Bewertungsmethoden der Reife und Stabilität zur Verfügung stehen. Schnell und einfach bezieht sich hierbei nicht auf die Methode an sich, sondern darauf, dass ein dritter die Bewertung bereits vorgenommen hat und auf diese zurückgegriffen wird. Zwei mögliche Ressourcen für solche Bewertungen sind:

- Andere Nutzer der Software, die als Referenz angeben werden. Falls andere Organisationen ähnliche Anforderungen haben oder sich sogar aktiv hinter das Projekt gestellt haben, ist dies zwar weiterhin eine eher informelle Bewertung, aber deshalb nicht we-

[34] z.B. in den Arbeiten Shah, S. (2006) und Roberts, J. et al. (2006)
[35] http://www.flossmetrics.org/
[36] http://www.sqo-oss.org/

niger zuverlässig. Referenzen finden sich entweder direkt auf der Projektseite, oder in Nachrichten von IT-Magazinen[37].

- Die Datenbank von OpenLogic[38]. Der Anbieter von Support und Beratung für OSS bietet kostenfreien Zugriff auf seine Datenbank, welche über 500 zertifizierte Open Source Projekte enthält. Die Zertifizierung ist ein langwieriger Prozess, der eine Vielzahl von Kriterien umfasst. Unter anderem: Lizenzüberprüfung, Projektstrukturanalyse (ähnliche wie sie oben vorgestellt wurde), Analyse aktueller Rechtsstreitigkeiten, Dokumentationsanalyse. Die komplette Zertifizierung umfasst 42 Schritte die in einem White-Paper erläutert werden[39].

Neben Stabilität und Reife der Community ist für eine Organisation insbesondere wichtig, ob Supportmöglichkeiten bestehen. Es bestehen prinzipiell folgende Supportmöglichkeiten:

- Die Organisation baut das Wissen intern auf. Dies bietet sich insbesondere an bei OSS, die relativ trivial zu nutzen ist und/oder quantitativ und qualitativ starke Kommunikationsplattformen (Mailingliste, Foren, etc.) vorzuweisen hat, wie es bei populären Projekten oft der Fall ist.
- Das OSS-Projekt wird von einem Unternehmen geleitet oder mitgetragen, das Supportverträge anbietet. Entgegen der öffentlichen Wahrnehmung ist dies, gerade bei großen Produkte, häufig der Fall. (Linux wird u.a. mitentwickelt von Red Hat und Novell, MySQL von Oracle, Eclipse von IBM. Dies deckt sich mit der Beobachtung, dass 42% aller Open Source-Entwickler für ihre Arbeit an OSS bezahlt werden[40].)
- Ein der Community fremder Dienstleister bietet Supportverträge an (z.B. listet typo3.org ca. 700 deutsche Dienstleister, die Support oder Beratung anbieten[41]).

Für eine Übersicht der Supportmöglichkeiten zu einem bestimmten OSS-Produkt ist die Projektseite selbstverständlich der erste Anlaufpunkt, bevor auf Nutzer- oder Entwicklermailing-

[37] z.B. www.heise.de, www.slashdot.org, www.t3n.de
[38] http://olex.openlogic.com/library/certified
[39] http://www2.openlogic.com/white-papers/openlogic-certification
[40] Jungwirth, C./Luthiger, B. (2007), S. 7
[41] http://typo3.org/community/service-offerings/

listen weitere Nachforschungen angestellt werden sollten[42]. Unabhängig von der gewählten Supportmöglichkeit ist es immer ratsam, eine gründliche Betrachtung der vorhanden Dokumentation durchzuführen. Eine aktuelle und umfangreiche Dokumentation kann schließlich nicht nur (womöglich kostenpflichtigen) Support sparen, sie ist auch ein Indiz für eine breit aufgestellte Community, die sich um die Nutzerbedürfnisse kümmert und sich der Tatsache bewusst ist, dass Dokumentation der Schlüssel zum produktiven Einsatz von Software ist.

3.2.2 Bewertung der Softwarequalität

Die Bewertung von Softwarequalität gestaltet sich oft schwierig, da Qualität eine Vielzahl von Aspekten umfasst (u.a. Sicherheit, Nutzerfreundlichkeit, Stabilität) und viele Produkte (auch im OSS-Bereich) einen Funktionsumfang erreicht haben, der eine komplette Überprüfung mit Tests praktisch unmöglich macht. Hinzu kommt, dass Aspekte wie Gebrauchstauglichkeit (die zwar beispielsweise in ISO 9241-110 definiert ist) und Nutzerkomfort nur eingeschränkt in absoluten Zahlen gemessen werden können, geschweige denn aus dem Quellcode ersichtlich sind. Jede Bewertung von Softwarequalität beschränkt sich deshalb auf wenige einzelne Qualitätsdimensionen. Trotzdem soll an dieser Stellle ein Überblick über einige Bewertungsmöglichkeiten gegeben werden.

Bei der Bewertung von Softwarequalität ist man bei proprietärer Software auf Zertifikate, historische Kennzahlen und Erfahrungen angewiesen. Bei OSS kann man zusätzlich den Quellcode selbst analysieren oder auf die Analyse Dritter zurückgreifen.

Zur Bewertung der Qualität kann daher aus den folgenden Herangehensweisen eine jeweils passende Kombination gewählt werden:

1. Zertifikate oder Klassifizierungen
    a. FIPS-140 - Kryptographische Tools, die von der US Regierung vorgeschriebene Anforderungen erfüllen, können zertifiziert werden[43].
    b. Coverity Scan Ladder - Das Sicherheitsunternehmen Coverity zertifiziert Programme nach einer detaillierten Quellcodeüberprüfung, u.a. auch ca. 300 OSS-Projekte[44]

---

[42] Zusätzlich ist natürlich eine Internetsuche nach "Produktname" + Support o.ä. immer eine Möglichkeit

[43] Vollständige Liste unter http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140val-all.htm

c. OpenLogic, ca. 500 OSS-Projekte sind zertifiziert (siehe Kapitel *3.2.1 Bewertung der Community*[45])

2. Einsatz der Software in Referenz-Umgebungen, bzw. besonders kritischen Umgebungen, die hohe Stabilität und Sicherheit erfordern.
   a. z.B. Einsatz von Linux in Siemens Magnet Resonanz Tomographen[46]
   b. Eine Vielzahl von Börsen nutzt Linux (z.B. in New York, London[47]), aber auch große Firmen und Behörden[48]
3. Das Heranziehen eines externen IT-Dienstleisters zur Bewertung der Softwarequalität. Software- oder Systemberatungsunternehmen können beauftragt werden, das Testen und Bewerten der Zuverlässigkeit eines OSS-Produktes durchzuführen. Dies verursacht selbstverständlich höhere Kosten als die Nutzung bestehender Bewertungen.
4. Die Analyse des Quellcodes zur Qualitätseinstufung mit Hilfe von Tools (genau wie die manuelle Community-Analyse ist dies jedoch nur bedingt aussagekräftig, also nur im Notfall zu nutzen):
   a. PMD[49] - Überprüft auf toten Code, typische Fehler, Code-Wiederholungen und Ressourcenverschwendung
   b. Checkstyle[50] - Überprüft, ob Quellcode einer gegeben Formatierungskonvention entspricht, was das Lesen und damit die Wartung vereinfacht. Ein gutes Abschneiden ist hier zumindest ein Indiz dafür, dass sich Projekt nicht nur auf die Funktionen, sondern auch auf die Qualität des Quellcodes konzentriert.

Allgemein gibt es die Behauptung, dass OSS unsicherer sei als CSS. Es wird argumentiert, dass die Offenheit des Quellcodes es böswilligen Programmierern erlaubt, leichter Sicherheitslücken zu finden, und, dass Freiwillige nicht in der selben Qualität programmieren wie bezahlte Programmierer. Dem entgegen steht einerseits die Tatsache, dass 42% aller Open

---

44 http://scan.coverity.com/all-projects.html
45 http://olex.openlogic.com/library/certified
46 http://linuxlookup.com/2006/sep/10/siemens_selects_suse_linux_software_system_to_ _mri_products
47 http://blogs.computerworld.com/london_stock_exchange_to_abandon_failed_windows_platform,
48 http://www.focus.com/fyi/50-places-linux-running-you-might-not-expect/
49 http://pmd.sourceforge.net/
50 http://checkstyle.sourceforge.net/

Source-Programmierer bereits für ihre Arbeit bezahlt wurden. Hinzu kommt, dass die Programmierer, die in ihrer Freizeit an einem Projekt arbeiten, aus persönlichen Gründen (sei es Spaß oder Lernwille) mindestens in gleichem Maße, oft aber sogar mehr, motiviert werden als durch Geld[51]. Studien bestätigen, dass OSS-Produkte teilweise sogar deutlich weniger Fehler enthalten, als ein durchschnittliches CSS-Produkt[52]. Es ist daher nicht verwunderlich, dass 79% aller Befragten angeben, dass ihre Erfahrung mit OSS[53] so gut sei, dass sie planen die Nutzung auszuweiten. Sicherheitslücken treten in OSS-Produkten weniger häufig auf als in CSS-Produkten und werden tendenziell schneller geschlossen. Natürlich sollte im Einzelfall trotzdem gewissenhaft geprüft werden, welche Qualität eine bestimmte OSS aufweist.

3.2.3 Checkliste

1. Ist die Community stabil und vertrauenswürdig genug, um zuverlässig langfristig planen zu können? Wie vertrauenswürdig ist die Quelle für diese Bewertungen?

Die Community um die betrachtete OSS-Groupware ist überaus stabil, wurde 2011 mit einem Community Award ausgezeichnet und wird durch ein Unternehmen unterstützt. Das Unternehmen bietet auch professionellen Support.

2. Ist die Software stabil und sicher genug für den Einsatz? Wie vertrauenswürdig ist die Quelle für diese Bewertungen?

Die Software wird von deutschen Bundesbehörden eingesetzt, einzelne Komponenten der Lösung (z.B. Postfix) sind von Coverity zertifiziert. Die Software gilt daher als ausreichend vertrauenswürdig.

[51] Jungwirth, C./Luthiger, B. (2007), S. 15

[52] Reasoning Inc (2006)

[53] Die Frage bezog sich auf OSS ohne Linux, im Anbetracht der als Beispiel genannten Referenzen, ist darf vermutet werden, dass die Zufriedenheit inkl. Linux noch höher wäre. aus CIO Insight OSS survey 2007 in http://www.groklaw.net/articlebasic.php?story=20070828132340846

3. Gibt es Referenzorganisation, welche die gleiche Software nutzen?

In welchem Rahmen die Software genau verwendet wird, ist zwar unklar, aber die Bundesbehörde tritt weiterhin stark für das Produkt ein, woraus zumindest zu schließen ist, dass ein Produktiveinsatz weiterhin vorliegt.

4. Gibt es ausreichende Dokumentation, Zugang zu den Kommunikationsplattformen der Community und professionellen Support?

Es existieren Handbücher, eine Wiki sowie eine User-Mailingliste. Zusätzlich gibt es professionellen Support von mehreren Unternehmen. Allein in Deutschland gibt es hierfür mehrere Anbieter.

Literaturempfehlung:

http://www2.openlogic.com/white-papers/openlogic-certification

Zur Einschätzung des OpenLogic-Zertifizierungsprozesses und einer Betrachtung der in Frage kommenden Faktoren.

3.3 Betrachtung der Total Cost of Ownership

Das folgende Kapitel soll die bereits beschriebene Softwareanalyse und die möglicherweise folgende Migration aus Sicht der Kostenplanung und -kontrolle beleuchten. Dieser Aspekt stellt selbstverständlich ein wesentliches Entscheidungskriterium für den Einsatz oder Nicht-Einsatz von OSS dar. Die ganzheitliche Kostenanalyse mit Hilfe des Total Cost of Ownership (TCO)-Ansatzes soll Entscheidungsträgern helfen, eine auch unter Kostengesichtspunkten valide Entscheidung zu treffen.

3.3.1 Einführung in TCO

Eines der wesentlichsten Entscheidungskriterien beim Erwerb eines Gutes ist die Gesamtheit der anfallende Kosten. Der Total Cost of Ownership-Ansatz (TCO) beschreibt ein Modell, das eine umfassende Betrachtung aller gegenwärtig und zukünftig anfallenden Kosten ermöglicht, die mit der Anschaffung und dem Betrieb von IT über den gesamten Lebenszyklus

hinweg verbunden sind.[54] 1987 definierte die Gartner Gruppe erstmals einen Ansatz zur ganzheitlichen Erfassung von Kosten und zeigte so eine neue Kostenbetrachtungsweise für Entscheidungsträger auf. Bisher wurden lediglich einmalige Anschaffungskosten und ggf. Wartungsverträge in Kostenkalkulationen aufgenommen worden, nicht aber indirekte und zukünftige Betriebskosten, was letztendlich Intransparenz und Ungenauigkeiten bei der Kalkulation zur Folge hatte.[55] Das Modell der Gartner Gruppe dient als Referenzmodell für Kostenbetrachtung, da es sich weltweit über die Jahre etabliert hat und sämtliche heutzutage relevanten Aspekte für eine Kostenanalyse in Betracht zieht.[56]

Der Einsatzbereich der TCO Rechnung kann sich über die gesamte IT Infrastruktur eines Unternehmens erstrecken. So können sämtliche relevante Komponenten wie (mobile) Endanwendergeräte, Mainframes, verteilte Systeme oder gesamte Infrastrukturinstallationen auf deren vollständige Anschaffungs- und Betriebskosten hin untersucht werden.
Ziel des Modells ist es, für alle Komponenten einer IT-Landschaft eine transparente und realistische (möglichst exakte) Kostenstruktur aufzustellen, welche Anschaffung und Betrieb (respektive Nutzung) mit in Betracht zieht. Diese Analyse soll als Basis für die Bewertung von Investitionsentscheidungen dienen.

Der TCO Ansatz hat seit der wachsenden Verbreitung und Bedeutung von Client Server Architekturen und verteilten Systemen an Bedeutung gewonnen, da die direkten Kosten (Anschaffungskosten) längst nicht mehr den größten Kostentreiber darstellen, vielmehr der Betrieb und die Nutzung (respektive Auslastung) dieser Systeme. [57]

3.3.2 Zweck von TCO Modellen

Die folgenden drei Elemente stellen im groben die Grundpfeiler eines Total Cost of Ownership-Modells dar. Diese erklären aus ihrer Natur heraus mit dieser Art der Kostenanalyse verbundenen Absichten.

[54] Riepl, L. (1998), S. 8
[55] Mieritz, L./Kirwin, B. (2005), S. 1 ff.
[56] Wild, M./Herges, S. (2000), S. 26
[57] Riepl, L. (1998), S. 8 ff.

Gesamtheit der Kostenbetrachtung:

TCO-Modelle betrachten erstmals neben den direkten Kosten, welche durch Rechnungen und Belege zugeordnet werden können, auch indirekte Kostenblöcke. Traditionelle Kostenbetrachtungen fokussieren sich lediglich auf besagte direkte Kosten, was zu einer nur bedingten Kostentransparenz führt. Durch die Detailtiefe der TCO Modelle und die daraus resultierende wesentlich exaktere Betrachtungsweise können die tatsächlichen Kosten der IT Infrastruktur ermittelt und so eine höhere Transparenz der Kostenstruktur geschaffen werden.[58]

Einbeziehung des Managements in den Entscheidungsprozess:

Investitionsentscheidungen und strategische IT-Entscheidungen werden zunehmend vom gehobenen Management getroffen und haben konsequenterweise einen Einfluss auf das gesamte Unternehmen. Die Anwendung von TCO-Konzepten zur Kostenbetrachtung ist ebenfalls darauf ausgelegt, dem Management in diesen Entscheidungen bestmöglich Hilfestellung zu leisten. Die Optimierung der Kostenstrukturen ist implizit als Aufgabe des Managements zu verstehen.[59]

TCO als Grundlage für Benchmarks:

Die ganzheitliche Betrachtung der IT-Infrastruktur erlaubt es Unternehmen, bezüglich ihrer Kostenstruktur Benchmarks durchzuführen, die unterschiedliche Produkte einander vergleichbar gegenüberstellen. Des Weiteren können unterschiedliche Strategien gegeneinander abgewogen werden, um so die optimale Ausrichtung für das Unternehmen zu identifizieren.

Durch die einheitliche Systematik der Kostenanalyse eignet sich der TCO Ansatz besonders für diese Art von Benchmarks.[60]

3.3.3 TCO Modell nach Gartner

Im Rahmen dieser Arbeit wird das TCO Modell nach Gartner betrachtet. Dieses Modell wurde in den 1980er Jahren als erstes seiner Art eingeführt, und wird seitdem stets verbessert und angepasst, was es de facto zum Industriestandard hat werden lassen.

---

[58] Riepl, L. (1998), S. 4

[59] Esterhazy, K./Schwab, W. (1998), S. 34

[60] Geissdoerfer, K./Gleich, R./Wald, A. (2009), S. 693 ff.

3.3.3.1 Grundlagen

Die ständige Anpassung des TCO-Ansatzes an neue Technologien am Markt führte zu einer diversifizierten Betrachtung und wir sprechen heute von einer Vielzahl an spezifizierten Modellen. Die Absicht und die Basisstruktur dieser Modelle ist jedoch bei sämtlichen Modellen ein und dieselbe. So werden sämtliche Kosten primär in direkte und indirekte Kosten gegliedert. Des Weiteren wurden vier Basisfaktoren definiert, welche diese Kosten wie folgt kategorisieren:

- originäre Aufgaben der EDV
- bestehende Infrastruktur
- Technischer Support
- IT-bezogene Verwaltung und Betrieb

Abbildung 2 zeigt die grundsätzliche Auffassung von Gartner bezüglich der Zusammensetzung der Gesamtbetriebskosten einer IT Infrastruktur.[61]

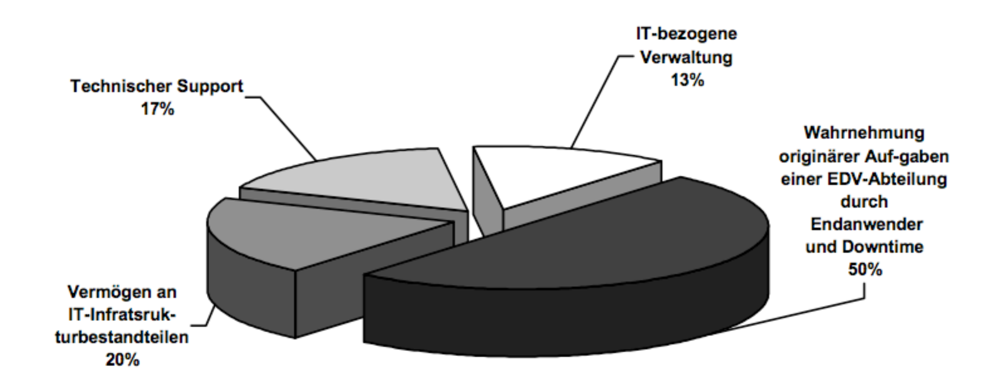


Abb. 2: Basisfaktoren der TCO nach Gartner[62]

3.3.3.2 Kostenstruktur des Modells

Direkte und indirekte Kosten einer IT Infrastruktur

Unter dem Begriff der direkten Kosten sind sämtliche Kosten zu verstehen, welche einer Abteilung bzw. Kostenträger direkt zugerechnet werden können.

---

[61] Vgl. Mieritz, L./Kirwin, B (2005), S. 1
[62] Riepl, L. (1998), S. 6

Diese Kosten (wie beispielsweise Abschreibungen für Hard- und Software, Leasinggebühren, Lizenzkosten, Löhne und Gehälter) sind in der Regel sehr genau bestimmbar, da sie in Form von Belegen (Rechnungen, Gehaltslisten etc.) dokumentiert sind.[63]

Wesentlich komplizierter verhält es sich mit den indirekten Kosten einer IT-Infrastruktur. So können zum Beispiel keinen unmittelbaren Mehrwert schaffende Vorgänge wie proaktive Störungsbehebung, Problemanalysen und Wartungsarbeiten als indirekte Kosten betrachtet werden, da nicht produktiv genutzte Arbeitszeit der Endanwender Kosten verursacht, denen kein unmittelbar anhand eines Beleges zu erfassender Wert gegenüber steht.[64]

Es ist folglich sehr schwer, diese indirekten Kosten zu messen, beziehungsweise sinnvoll zu quantifizieren. Der standardisierte TCO Ansatz jedoch bezieht diese Kosten in die Kalkulation mit ein.

Nachfolgend werden weitere Kostenarten beschrieben, welche großen Einfluss auf die Höhe der TCO haben.

Nutzung von Hard- und Software

Hardwarekosten sind in erster Linie Aufwendungen, die aus der Anschaffung und Nutzung der Hardware resultieren. Darunter fallen ebenfalls Upgrades und Ersatzteile. Dabei spielt es keine Rolle, ob die Hardware Zugriffsbereich des Endanwenders oder in der dahinterliegenden Infrastruktur eingesetzt wird. Die Kosten sind indifferent zu betrachten. Hardwarekosten sind in der Regel relativ einfach zu erfassen, da es sich bei Hardware um physikalische Einheiten handelt, die einfach zu zählen und quantifizieren sind. Außerdem sind Neuinvestitionen in diesem Bereich oft weit im Voraus abschätzbar.[65]

Ist in einem Unternehmen beispielsweise der Archivierungsserver-Speicher zu 98% ausgelastet, liegt es auf der Hand, dass zusätzliche Speichereinheiten benötigt werden um einen reibungslosen operativen Betrieb zu gewährleisten.

[63] Vgl. Wild, M./Herges, S. (2000), S. 11
[64] Vgl. Krcmar, H. (2005), S. 182
[65] Vgl. Wild, M./Herges, S. (2000), S. 12

Analog dazu verhalten sich Anschaffungs- und Nutzungskosten von proprietären Softwareprodukten in Unternehmen. Allerdings müssen gemäß der Natur von Softwareprodukten, welche an Lizenzmodelle und weitere Bedingungen geknüpft sind, eine Vielzahl an zusätzlichen Kosten berücksichtigt werden, um eine möglichst exakte Bewertung dieser Kosten vornehmen zu können.

Darüber hinaus können komplexe Abhängigkeiten zwischen verschiedenen Softwareprodukten (Betriebssystem – Endanwendung) die Kostenermittlung erschweren.

Betrieb und Verwaltung

In die Kostenkategorie "Betrieb und Verwaltung" fallen sämtliche Aufwendungen, welche für jegliche Form der unmittelbaren Betriebserhaltung entstehen. Dies sind vor allem Personal- und Wartungskosten sowie IT bezogene Dienstleistungen, die einem reibungslosen Betrieb der IT Infrastruktur dienen.

Diese Kostenkategorie fasst sämtliche betriebliche Aufwendungen zusammen, welche administrativer Natur (in diesem Falle bezogen auf die IT) sind. Dies können unter andrem Mieten für Rechenzentrumsräumlichkeiten, aber auch Kosten für Schulungen des IT-Personals sein. [66]

Systemausfälle

Die aus System-Downtime bzw. IT bezogenen Ausfälle resultierenden Kosten werden unter dieser Kategorie zusammengefasst. Unter dem Begriff der Downtime ist die Zeit zu verstehen, in der Mitarbeiter die zur Wahrnehmung ihrer Aufgaben erforderlichen Systeme einer IT Infrastruktur nicht nutzen können und dementsprechend zur Unproduktivität gezwungen sind. Zur Berechnung wird in der Regel zunächst diese Produktivitätsminderung (ausgedrückt in Zeiteinheiten) ermittelt und dann mit einer Kennzahl gewichtet, welche den Stundensatz des jeweiligen Mitarbeiters reflektiert. Ein zweiter Ansatz von Gartner, diese Downtime-Kosten zu erfassen, basiert auf der Interpretation dieser Kosten als zukünftiger Umsatzverlust. [67]

[66] Vgl. Wild, M./Herges, S. (2000), S. 13
[67] Vgl. ebd. S.14

3.3.4 Maßnahmen für die Erhebung der TCO im Zuge einer Softwaremigration

Das folgende Kapitel soll dem Leser ein methodisches Vorgehen an die Hand geben, welches es ermöglicht, unter strategischen Gesichtspunkten eine TCO-Analyse praktisch anzuwenden, um diverse Softwarestrategien gegeneinander abzuwägen. Teil dieser Analyse soll ein internes Benchmarking (wie in Kapitel *3.3.2 Zweck von TCO Modellen* theoretisch erläutert) sein, um unterschiedliche Softwarelösungen direkt vergleichen zu können. Es sollen hierbei vor allem die Besonderheiten von OSS bezüglich der Kostenstruktur hervorgehoben werden.

Die nachfolgende Tabelle 3 stellt eine Checkliste dar, die mögliche zu erhebende Kosten aufzeigt. Wie diese Kosten innerhalb der jeweiligen Organisation erhoben werden, kann an dieser Stelle nicht einheitlich aufgezeigt werden, da sich die Arbeitsstrukturen (je nach Branche, Größe usw.) stark unterscheiden und eine solche Betrachtung den Rahmen dieses Ratgebers sprengen würde.

Zu allererst sollte man beachten, dass bereits für eine solche Evaluation selbst Kosten anfallen. Die Sinnhaftigkeit einer ausführlichen TCO-Studie muss daher von Fall zu Fall vorab entschieden werden. Vor allem wenn es sich um kleinere Implementierungen handelt, bei denen proprietäre Software (mit sehr geringen Lizenzkosten) gegen OSS abgewägt wird, können die Evaluationskosten die Anschaffungskosten der kommerziellen Software bereits überschreiten.

Die folgenden Tabellen 3 und 4 stellen zwei beispielhafte Erhebungen und Kalkulationen gemäß dem TCO-Ansatz dar. Dieser soll bei der Entscheidung für die zukünftigen Softwarestrategie helfen. Verglichen werden ein Update der bisherigen proprietären Lösung und die Migration auf eine Open Source-Groupware. Hierbei dienen abermals die Ausgangswerte des Beispielunternehmens im Ratgeber als Basis für die Kostenkalkulation.

Literaturempfehlung:

Total Cost of Ownership (TCO) – Ein Überblick:

Beschäftigt sich mit Grundlagen und praktischen Empfehlungen.

http://geb.uni-giessen.de/geb/volltexte/2004/1577/pdf/Apap_WI_2000_01.pdf

Praxisbeispiele	Mögliche Kostenstellen	Beispielkalkulation proprietäre Software	Beispielkalkulation Open Source Software
Direkte Kosten			
Hardware Serverkosten	Belege und Rechnungen im Rechenzentrum, Abteilungen, Einkauf	8500 € (2 Server-/ Storageeinheiten)	8500 € (2 Server-/ Storageeinheiten)
Software Lizenzkosten	Belege und Rechnung von Softwarekäufen und Software Lizenzverträgen im Rechenzentrum, Abteilungen, Einkauf. Hierbei muss auf die vertraglich festgelegten Zahlungsbedingungen geachtet werden (monatliche/ einmalige Lizenzkosten)	28200€ bisherige Gesamtlizenzkosten. 1500 € Lizenzgebühr für den Mailserver. 1200 € Lizenzgebühr für das Serverbetriebssystem 25000€ für Update der 250 Clients á 100€ Lizenzkosten.	Lizezenzkosten fallen für diese Software nicht mehr an es kann also mit Kosten Einsparungen in Höhe von 28200 € gerechnet werden.
Wartung und Support	Diese Kosten können in der Regel mit Hilfe des Controlling oder des Einkaufs ermittelt werden.	Bestehende Verträge mit Partnern in Höhe von jährlich 780 €.	Neue abgeschlossene Wartungsverträge für Open Source: 1000€ Der Anbieter der wesentlich teureren Supportleistung, garantierte schnellere Reaktionszeiten und häufigere Wartungsintervalle, was sich positiv auf die Systemausfallzeiten auswirkt.
IT Löhne und Gehälter	Diese Kosten lassen sich auf Gehaltsabrechnungen in entsprechenden Personal-/ Finanzabteilungeneinsehen.	80000€ Jahresgehalt des IT Mitarbeiters.	80000€ Jahresgehalt des IT Mitarbeiters.

Externe Dienstleister	Belege und Rechnungen aus dem Einkauf und entsprechenden Abteilungen welche relevante Dienstleister eingekauft haben.	Externer Dienstleister übernimmt das Update des bisherigen Systems. Hierfür fallen 850€ an.	Consultant für Migration wird in den Projektkosten für Migration miteinberechnet.
Indirekte Kosten			
Opportunitätskosten für Systemausfälle (Downtime, Wartung)	Wie bereits erwähnt können diese Kosten nur schwer ermittelt werden. Systemausfälle in Stunden können auf Basis der Personalkosten quantifiziert werden.	Im letzten Jahr fiel das System für insgesamt 12 Stunden aus. Dies verursachte Kosten in Höhe von geschätzten 80.000€. In den folgeperioden wir auf Grund des Updates eine höhere Verfügbarkeit erwartet. Die Ausfallkossten werden um 10000€ auf 70000€ gesenkt. (Verlust durch einen Arbeitstag bei 60% Produktivität ohne IT Zugriff)	Systemausfälle sollen durch die Migration verringert werden. Hierfür stehen aber keine Anhaltspunkte zur Verfügung. Im Zuge der möglichst exakten Erhebung der Kosten wird der Selbe Betrag von 80000€ angesetzt. In den Folgeperioden wird auf Grund des neuen Systems und des verbesserten Suppports mit weniger Ausfallzeiten gerechnet. Es wird ein kalkulatorisches Wagnis von 60000€ veranschlagt.
Versicherungen	Bestehende Verträge für Versicherungen mit Bezug zur IT Versicherungen (Immobilien/ IT-Hardware/ Systemverfügbarkeit)	Versicherungspolicen von jährlich 150 € sind für die IT fällig.	Versicherungspolicen von jährlich 150 € sind für die IT fällig.
Umlage Verwaltungskosten für IT	Keine exakte Umlage möglich. Pauschaler Ansatz eines prozentualen Anteils der Gesamtverwaltungskosten, je nach Größe der IT.	60	60

Umlage Raumkosten (Miete) des Rechenzentrum	Quadratmeterpreis der Rechenzentrum/ IT-Räumlichkeiten	Das Rechenzentrum der Firma hat eine Grundfläche von 20qm im Industriegebiet der ansässigen Stadt beträgt der Quadratmeterpreis 80€. Es fallen also Gesamtraumkosten von 1600€ an.	Das Rechenzentrum der Firma hat eine Grundfläche von 20qm im Industriegebiet der ansässigen Stadt beträgt der Quadratmeterpreis 80€. Es fallen also Gesamtraumkosten von 1600€ an.
Umlage für Energie und Kühlungskosten	Verbrauchsabhängige Kalkulation der Energiekosten.	Server und Kühlung 2000 KW bei 0,17€ pro KW/h. 3000€	Server und Kühlung 2000 KW bei 0,17€ pro KW/h. 3000€
Sonstige Kosten			
Migrationskosten		Es fallen keine Migrationskosten an, da die bisherige Software beibehalten wird.	Projektkosten werden auf 7200 € geschätzt.
Schulungskosten	Schulungskosten für neue Software vom Herausgeber oder externen Schulungsdienstleistern	Es fallen keine Schulungskosten an, da die bisherige Software beibehalten wird.	Das Unternehmen investiert 5000€ in die Entwicklung eines eLearnings zur Schulung der Mitarbeiter auf dem neuen System, welches 3 Wochen vor dem Übergang bereitgestellt wird.
Ggf. weitere Kosten für Migration	Evtl. -Vertragsstrafen		

Tabelle 3: Gegenüberstellung der Kostenerhebung für proprietäre und Open Source Lösungen

Total Cost of Ownership Kalkulation	TCO proprietäre Lösung			TCO Open Source Lösung		
	Jahr 1	Jahr 2	Jahr 3	Jahr 1	Jahr 2	Jahr 3
Direkte Kosten	**118.330 €**	**80.780 €**	**80.780 €**	**89.500 €**	**81.000 €**	**81.000 €**
Hardware Kosten	8.500 €	0 €	0 €	8.500 €	0 €	0 €
Software Lizenzkosten	28.200 €	0 €	0 €	0 €	0 €	0 €
Wartung- und Supportverträge	780 €	780 €	780 €	1.000 €	1.000 €	1.000 €
IT Löhne und Gehälter	80.000 €	80.000 €	80.000 €	80.000 €	80.000 €	80.000 €
Externe Dienstleister	850 €	0 €	0 €	0 €	0 €	0 €
Indirekte Kosten	**84.810 €**	**74.810 €**	**74.810 €**	**84.810 €**	**64.810 €**	**64.810 €**
Opportunitätskosten für Systemausfälle	80.000 €	70.000 €	70.000 €	80.000 €	60.000 €	60.000 €
Versicherungen	150 €	150 €	150 €	150 €	150 €	150 €
Umlage IT Verwaltungskosten	60 €	60 €	60 €	60 €	60 €	60 €
Umlage Raumkosten (Miete) des Rechenzentrum	1.600 €	1.600 €	1.600 €	1.600 €	1.600 €	1.600 €
Umlage für Energie und Kühlung	3.000 €	3.000 €	3.000 €	3.000 €	3.000 €	3.000 €
Sonstige Kosten	**0 €**	**0 €**	**0 €**	**12.200 €**	**0 €**	**0 €**
Migrationskosten	0 €	0 €	0 €	7.200 €	0 €	0 €
Schulungskosten	0 €	0 €	0 €	5.000 €	0 €	0 €
Ggf. weiter Kosten für Migration	0 €	0 €	0 €	0 €	0 €	0 €
Summe	**203.140 €**	**155.590 €**	**155.590 €**	**181.510 €**	**145.810 €**	**145.810 €**
TCO			**514.320 €**			**473.130 €**
Einsparungen:						**41.190 €**

Tabelle 4: Beispielkalkulation der TCO

4 Transformationsphase

Nachdem im Verlauf der Evaluationsphase analysiert wurde, welche Anforderungsabdeckung und Kostenstruktur einzelne Software-Lösungen mit sich bringen und die Entscheidung auf eine der Lösungen gefallen ist, gilt es im Verlauf der Transformationsphase, die Einführung der Software zu planen und durchzuführen. Die Einführung besteht dabei aus der konkreten (technischen) Migration sowie der Dokumentation und Anwenderschulung.

4.1 Migration und Rollout

Hat eine Organisation, unter Berücksichtigung der in den bisherigen Kapiteln aufgeführten Punkte, den Bedarf ermittelt, eine verwendete proprietäre Anwendung durch eine neue Applikation aus dem Open Source-Umfeld zu ersetzen, so gilt es, diese Umstellung genau zu planen, um einen reibungslosen Ablauf gewährleisten zu können. Das folgende Kapitel wird aus diesem Grund verschiedene Migrationsstrategien erläutern und mögliche Fallstricke aufzeigen. Es sei an dieser Stelle darauf hingewiesen, dass dieses Kapitel nicht den Anspruch erhebt, alle Einzelheiten einer Migration darzulegen. Am Ende dieses Kapitels finden Sie entsprechende weiterführende Literatur, die für einen tieferen Einblick in die Thematik herangezogen werden kann. Ziel der Darstellung im Rahmen dieses Ratgebers ist es, einen Überblick über die Thematik zu vermitteln.

4.1.1 Definition

Sneed, Wolf und Heilmann definieren Softwaremigration wie folgt:

> *Softwaremigration bezeichnet die Überführung eines Softwaresystems in eine andere Zielumgebung oder in eine sonstige andere Form, wobei die fachliche Funktionalität unverändert bleibt.*[68]

Es ist festzustellen, dass Migration lediglich die Aufgabe hat, ein Alt-System (samt Daten) in ein neue Umgebung zu überführen und die gleiche Funktionalität zu gewährleisten. Es ist keine Verbesserung der Qualität (Software Reengineering) damit gemeint und explizit auch kein Hinzufügen von neuen Funktionen. Beim Einsatz von Standardsoftware hat man natür-

[68] Sneed, H./Wolf, E./Heilmann H. (2010), S. 25

lich keinen oder zumindest nur einen beschränkten Einfluss auf die Funktionen, welche implementiert werden. Somit kann es durchaus sein, dass bei der Migration von einer älteren Version auf eine neuere, bei der keine Abwärtskompatibilität besteht und somit eine Migration erforderlich ist, nach der Migration neue Funktionen vorhanden sein werden.

Wird eine neue Software eingeführt, kann dies bedeuten, dass die bestehende und eingesetzte Produktlinie beibehalten wird. Wird trotzdem eine Migration nötig, da keine Abwärtskompatibilität gewährleistet ist, so spricht man von einer *fortführenden Migration*. Wird die Produktlinie bei der Einführung hingegen gewechselt, so nennt man diese Art von Migration *ablösende Migration*.

Neben der Migration eines Altsystems oder einer Altanwendung (auf deren Ausprägungen wird in Kapitel *4.1.2 Migrationsarten* näher eingegangen) gibt es noch die Möglichkeit der Neuentwicklung des Systems sowie den Einsatz von Standardsoftware, soweit noch nicht eingesetzt. Abbildung 3 zeigt hierbei, dass lediglich bei einer Migration die gleiche Funktionalität auch im neuen System verfügbar ist. Bei den Alternativen der Neuentwicklung und dem Einsatz von Standardsoftware ändert sich der Funktionsumfang meist dahingehend, dass neue Funktionen hinzugefügt werden. Diese Alternativen werden in diesem Kapitel thematisch nicht behandelt.

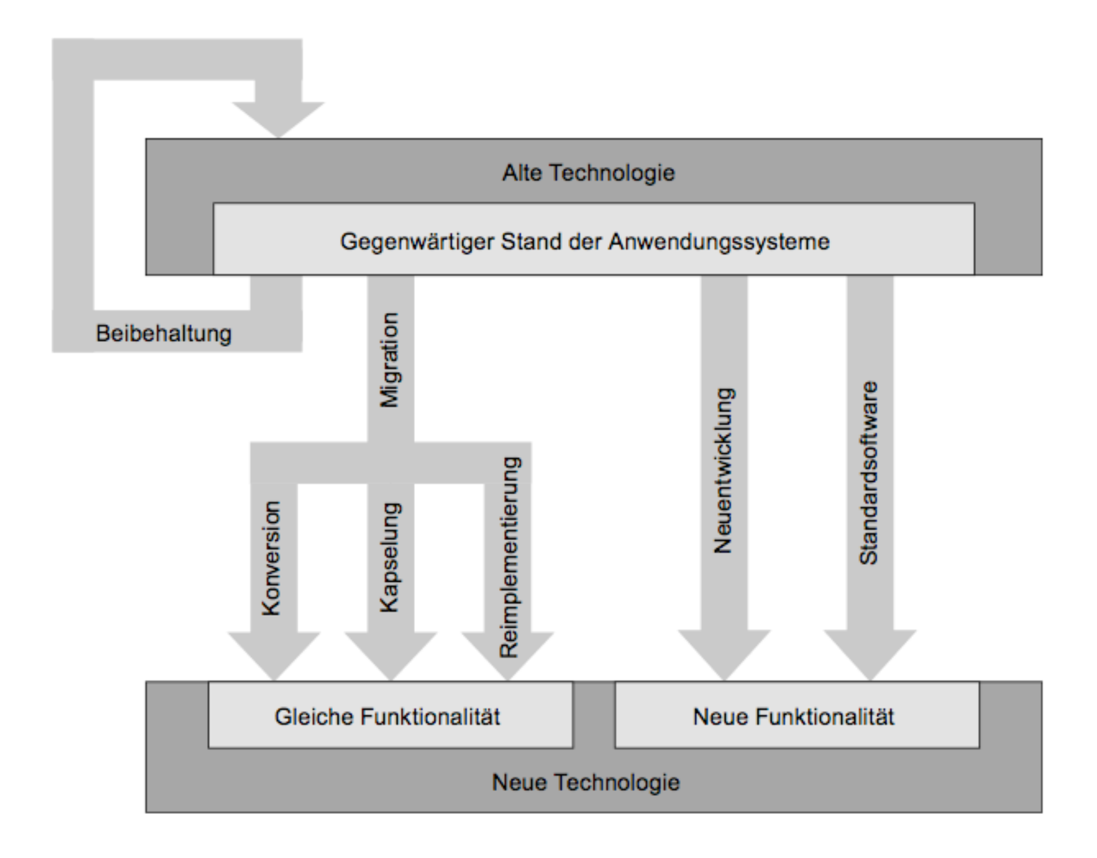


Abb. 3: Mögliche Migrationspfade und Alternativen[69]

4.1.2 Migrationsarten

Das folgende Kapitel beschreibt vier unterschiedliche Migrationsarten. Diese stellen mögliche Teilmigrationen dar, welche in einen System durchgeführt werden können. Hierzu muss das System oder die Anwendung zerlegbar sein. Ist dies nicht der Fall, so kann es unter Umständen schwierig sein, die unterschiedlichen Migrationsarten strikt zu trennen und durchzuführen. Soll dies trotzdem geschehen, sind zunächst Sanierungsarbeiten notwendig. Unter Sanierungsarbeiten wird hierbei das sogenannte *Reengineering* verstanden. Dies hat zum Ziel, beispielsweise eine bessere Performance oder Wartbarkeit zu erzielen. In der Gesamtbetrachtung soll durch eine Sanierung die Qualität des Systems bzw. der Anwendung gesteigert werden[70]. In dem vorliegenden Kontext würde dies bedeuten, dass eine Zerlegbarkeit des Systems hergestellt wird.

[69] Sneed, H/Wolf, E./Heilmann H. (2010), S. 11

[70] Vgl. ebd., S. 28

Datenmigration

Von einer Datenmigration spricht man, wenn Daten von einem „Datenhaltungs- bzw. Datenbanksystem auf derselben Plattform versetzt werden oder [...] mit unveränderter Datenhaltung auf eine andere Plattform übertragen werden“[71]. Das eigentliche System bzw. Programm wird in seiner gegenwärtigen Form beibehalten (wenn auch möglicherweise die Zugriffspfade auf die Daten angepasst werden müssen). Dies ist beispielsweise bei einer Migration von hierarchischen Datenbanksystemen zu relationalen Datenbanksystemen oder von relationalen Datenbanksystemen zu objektorientierten Datenbanksystemen der Fall.

Der Aufwand bei einer Datenmigration kann sehr unterschiedlich sein. So kann eine Migration von einem relationalen Database Management System (DBMS) wie Oracle zu einem ebenfalls relationalen DBMS wie DB2 fast vollständig automatisiert werden und der Aufwand dadurch recht gering gehalten werden. Anders gestaltet sich die Situation bei einer Migration von einer hierarchischen Datenbank zu einer relationalen. Hierbei handelt es sich um zwei unterschiedliche Technologien und Grundtypen, was eine Automatisierung erschwert oder gar nicht erst ermöglicht. Darüber hinaus müssen Anpassungen an der Zugriffslogik in den jeweiligen Anwendungen berücksichtigt werden, wenn Migrationen von unterschiedlichen Technologien von Statten gehen.

Programm-Migration

Bei einer Programm-Migration handelt es sich um eine Änderung an der Anwendungslogik. Die Daten bleiben unverändert. Programme „können in eine andere Sprache in der gleichen Umgebung, in derselben Sprache in eine andere Umgebung oder in einer anderen Sprache und in eine andere Umgebung migriert werden“[72]. Um eine andere Sprache verwenden zu können, muss der Code entweder konvertiert oder aber zumindest gekapselt werden, (abhängig von der gewählten Migrationsstrategie).

---

[71] Sneed, H/Wolf, E./Heilmann H. (2010), S. 40
[72] ebd., S. 41

Benutzerschnittstellen-Migration

Bei der Migration einer Benutzerschnittstelle werden weder Daten, noch die Anwendungslogik tangiert. Es handelt sich hierbei lediglich um eine Anpassung, bzw. einen Wechsel der Präsentationsschicht. Gründe hierfür sind meist der Wunsch nach Verbesserung der Benutzerfreundlichkeit oder Integration mehrerer Systeme unter einer einheitlichen Oberfläche.

Systemschnittstellen-Migration

Zu einer Migration von Systemschnittstellen kommt es immer dann, wenn sich etwas in der Umgebung[73] des Systems verändert hat. Wenn z.B. ein System innerhalb der Organisation migriert wurde und nun ein moderneres Schnittstellenprotokoll verwendet, so müssen die Schnittstellen aller anderen Anwendungen zu diesem System ebenfalls auf die neuen Gegebenheiten angepasst und migriert werden.[74] Nur durch eine Anpassung der Schnittstellen kann die Interoperabilität, zwischen den System auch nach der Migration eines Bestandteils weiter gewährleistet werden. Es ist daher bei der Migration eines Systems von enormer Wichtigkeit, dessen Umfeld zu berücksichtigen.

4.1.3 Migrationsstrategien

Wie in Abbildung 3 zu sehen, gibt es drei denkbare Migrationsstrategien. Diese sind zum einen die *Reimplementierung*, die *Konversion* und die *Kapselung*. Die Unterschiede werden im Folgenden kurz erläutert.

Reimplementierung

Bei der *Reimplementierung* einer Anwendung bzw. eines Systems wird das Altsystem in einer anderen Programmiersprache neu entwickelt. Es handelt sich hierbei um eine 1:1-Neukodierung des System mit exakt den selben Funktionen und auch der selben Architektur

[73] Umgebung wird hier als die technische Infrastruktur verstanden. Hier können Abhängigkeiten bestehen, welche bei Änderungen an einem entfernten System auch Änderungsbedarf an dem betreffenden System bedeuten.

[74] Vgl. Sneed, H/Wolf, E./Heilmann H. (2010), S. 43

(nur der Code ändert sich). Wenn man von Reimplementierung spricht, ist meist die händische Neuerstellung gemeint und keine automatisierte Konvertierung.[75]

Konversion

Bei der Konversion wird im Prinzip analog zur Reimplementierung vorgegangen. Der grundlegende Unterschied besteht darin, dass die Konversion auf einer automatisierten Übertragung des Altsystems in eine neue Umgebung beruht. Es werden also Werkzeuge verwendet, welche die Code-Basis sowie die Daten in ein neues Format bringen. Bei dieser Strategie gilt es jedoch zu beachten, dass jede Konversion nur so gut sein kann, wie die Werkzeuge, die für sie verwendet werden. Diese zu entwickeln, kann sehr aufwendig und somit teuer sein. Es sollte deshalb genauestens abgewägt werden, ob eine Konversion im Einzelfall Sinn ergibt. Dies ist meist nur dann der Fall, wenn man eine große zu konvertierende Code-Basis hat.

Kapselung (Wrapping)

Bei der *Kapselung* bleibt das Altsystem in seiner Gänze erhalten. Lediglich Schnittstellen werden nach außen neu definiert und integriert, damit das Altsystem mit einer es umgebenden Softwarehülle in der gewünschten neuen Programmiersprache kommunizieren kann. Diese Hülle hat keine zusätzlichen Funktionen. Sie dient lediglich zur Verwendung neuer Technologien. Beispielsweise kann eine COBOL-Anwendung durch eine Java-Anwendung gekapselt werden um diese anderen Systemen objektorientiert zugänglich zu machen.

4.1.4 Migrationsvorbereitungen

Messung

Vor Beginn einer jeden Migration ist es unabdingbar, dass das zu migrierende Objekt vermessen wird. Vermessen heißt in diesem Kontext, Kennzahlen wie Größe, Komplexität und Qualität des Objektes zu bestimmen.[76] Erst mit diesen Daten ist es möglich, im Anschluss eine Aufwandschätzung durchzuführen und eine Entscheidung über die Migration zu treffen. Zunächst erfolgt die Messung der Größe. Diese kann anhand unterschiedlichster Kriterien vorgenommen werden. Die einfachste und zugleich die ungenaueste Vorgehensweise ist hierbei

[75] Vgl. Sneed, H/Wolf, E./Heilmann H. (2010), S.11 u. S. 35
[76] Vgl. ebd., S. 134

das simple Zählen von Codezeilen. Ungenau ist diese Variante aus dem Grund, dass viele Zeilen aus einfachen Kommentaren oder sehr trivialen Operationen bestehen, die eigentlich keine Last auf dem System erzeugen. Besser ist die Bestimmung der Anzahl von Anweisungen. Weitere Messkriterien sind:

- **Function Points:** Systemschnittstellen nach außen sowie interne Dateien und Datenbanken
- **Objects Points:** Umfang des Objektmodells
- **Data Points:** Größe der Datenbank

Laut Sneed sollte man „in jedem Fall [...] für die Planung der Migration wissen, wie viele [...] Klassen oder Module, Dateien, Datenbanktabellen, Benutzer- und Systemschnittstellen und wie viele Jobsteuerungsprozeduren bzw. Jobskripte es gibt“[77].

Nach Ermittlung der Größe gilt es, die Komplexität des Systems bzw. der Anwendung zu bestimmen. Die Komplexität bei Software ist mehrdimensional und es gilt, unterschiedliche Komplexitäten (z.B. die der Datenstruktur, Datenschnittstellen, Architektur, der Sprache selbst oder der Verarbeitungslogik) auf einen gemeinsamen Nenner, nämlich eine Gesamtkomplexität zu bringen.

Letztlich muss noch die Qualität eines bestehenden Systems bestimmt werden. Hauptkriterien hierfür sind *Modularität*, *Portabilität*, *Konvertierbarkeit* und *Wiederverwendbarkeit.* Diese Kriterien können in Kürze wie folgt beschrieben werden:

- **Modularität:** Größe einzelner Bausteine sowie Kopplungen zwischen ihnen
- **Portabilität:** Verhältnis der umgebungsspezifischen Anweisungen zur Gesamtanzahl der Anweisungen
- **Konvertierbarkeit:** Anzahl vollständig konvertierbarer Anweisungen zur Gesamtanzahl der Anweisungen
- **Wiederverwendbarkeit:** Anzahl der Codeabschnitte, die in einem anderen Kontext unverändert wiederverwendet werden können

[77] Sneed, H/Wolf, E./Heilmann H. (2010), S. 135

Systembewertung

Anhand der eben beschriebenen Messwerte kann im Folgenden eine Bewertung des Systems durchgeführt werden. Messwerte wie die Komplexität und der jeweilige Erfüllungsgrad einzelner Qualitätskriterien können mit einem Wert zwischen 0 und 1 angegeben werden. Die Aufschlüsselung gestaltet sich dabei wie folgt:

Von	Bis	Bewertung
0,8	1	Ausgezeichnet
0,6	0,8	Gut
0,4	0,6	Ausreichend
0	0,4	Schlecht

Tabelle 5: Softwarebewertungsskala[78]

Ähnlich der Portfolioanalyse, kann ein System mit den ermittelten Kennzahlen bewertet werden. Dazu ist es abschließend noch notwendig, die Wichtigkeit des Systems für das Geschäft zu bestimmen. Hierbei kann ein System als

- geschäftskritisch (0,8 bis 1),
- sehr wichtig (0,6 bis 0,7),
- mäßig wichtig (0,1 bis 0,5) oder
- unwichtig (0 bis 0,1)

eingestuft werden. Die quantitative Bewertung wird gemäß ISO 9126[79] vorgenommen. Laut dieser Norm können Systeme in einem Koordinatensystem abgetragen werden. Auf der Ordinate wird hierbei die technische Qualität und auf der Abszisse die relative betriebswirtschaftliche Bedeutung abgetragen.

Aufwandsschätzung

Nachdem nun Messung und Bewertung des Systems durchgeführt sind, kann eine quantitative Aufwandsschätzung vorgenommen werden. Diese ist wichtig, um den Aufwand und damit die Kosten für eine Migration zu bestimmen und wirtschaftliche Überlegungen anzustellen:

---

[78] Vgl. Sneed, H/Wolf, E./Heilmann H. (2010) nach ISO 9126, S. 137

[79] Vgl. ebd.

Lohnt sich eine solche Migration überhaupt? Kann/soll sie im eigenen Hause durchgeführt werden oder soll sie an einen externen Dienstleister weitergegeben werden? Ist letzteres der Fall, so kann die Aufwandsschätzung auch dazu beitragen, unterschiedliche externe Angebote zu überprüfen und auszuwerten.

Für die Aufwandschätzung einer Codemigration gibt es verschiedene Verfahren, welche im Detail der am Ende dieses Kapitels empfohlenen Literatur entnommen werden können. Beispielhaft soll an dieser Stelle nur kurz auf die *Aufwandsbestimmung mit Object Points* eingegangen werden. Dabei werden Punkte für unterschiedliche Bestandteile eines Systems vergeben. So wird eine Klasse bspw. mit 4 Punkten, eine Methode mit 3 Punkten, eine Assoziation mit 2 und ein Attribut mit einem Punkt bewertet. Durch diese Kriterien und einiger Rechnungen ist es möglich, den Aufwand abzuschätzen.

Zur Aufwandsschätzung bei Datenmigrationen kann exemplarisch die Data-Point-Methode herangezogen werden. Bei dieser Methode werden für verschiedene Eigenschaften der Datenbank Punkte vergeben. So werden z.B. für

- eine Tabelle oder Satzart = 4 Data Points,
- einen Schlüssel = 2 Data Points,
- einen Querverweis bzw. eine Beziehung = 2 Data Points und für
- ein Attribut = 1 Data Point

vergeben. Die Summe dieser Punkte wird anschließend durch die mittlere Produktivität für Datenbanken vergleichbarer Größe geteilt.

Mitarbeiterinformation

Neben den beschriebenen Punkten ist es besonders wichtig, auch die Endbenutzer in den Migrationsprozess einzubinden. Das bedeutet, dass die Anwender schon frühzeitig über eine Umstellung informiert werden und die Vorteile für sie klar herausgestellt werden. Durch derartige Maßnahmen kann die Angst genommen und die Akzeptanz gesteigert werden. Während der Migration sollten die Mitarbeiter über den aktuellen Status auf dem Laufenden gehalten werden.

4.1.5 Migrationsansätze

Es gibt eine Vielzahl von Migrationsansätzen, welche verwendet werden können. Hier sollen jedoch lediglich zwei Ansätze exemplarisch aufgezeigt werden. Denkbare weitere Ansätze wären die so genannten *Butterflymigration*, der *Renaissance-Migrationsprozess*, das *COREM-Verfahren*, der *SMART-Ansatz* und die *objektorientierte Migration*. All diese Verfahren können im Detail der Literatur entnommen werden.

Chicken-Little

Der sogenannte *Chicken-Little-Ansatz* beschreibt einen von der University of California in Berkeley entwickelten Ansatz. Dieser ist wenig risikobehaftet, da er eine sanfte und in (elf) Schritten vorzunehmende Migration vorsieht[80]. Diese Schritte können im Wesentlichen wie folgt kategorisiert werden.

- Schritte 1 und 2: Analyse und Zerlegung des Altsystems
- Schritte 3 bis 5: Definition des Zielsystems
- Schritte 6 und 7: Einrichtung des Zielsystems
- Schritte 8 bis 10: Migration des Altsystems
- Schritt 11: Umstellung

Während des gesamten Migrationsprozesses bestehen das alte und neue System parallel. Einen großen Teil des Migrationsprozesses machen Analyse und Entwurf aus.

Da es sich beim Chicken-Littel-Ansatz um eine inkrementelle Migration handelt, funktioniert diese am besten bei einem zerlegbaren System. Bei semi-zerlegbaren und nicht zerlegbaren Systemen sind *Gateways* notwendig, welche die einzelnen Funktionen voneinander trennen. Je weniger zerlegbar ein System ist, desto mehr *Gateways* werden benötigt und umso mehr steigt auch der Aufwand für eine solche Migration. Diese *Gateways* sind Brücken zwischen dem alten und dem neuen System. Sie haben die Aufgabe, Zugriffe auf bestimmte Funktionen oder Daten des Neusystems aus dem Altsystem heraus zu ermöglichen. So kann ein *Gateway* beispielsweise einer Funktion im Altsystem den Zugriff auf die migrierte neue Datenbasis im Neusystem ermöglichen.

[80] Vgl. Sneed, H/Wolf, E./Heilmann H. (2010), S. 50

Der *Chicken-Little-Ansatz* ist ein von Vorsicht und Risikominimierung geprägter Ansatz. Durch das schrittweise Vorgehen werden ein niedriges Risiko, eine hohe Flexibilität und die Aufrechterhaltung des Systembetriebes erreicht. Nachteilig wirken sich hingegen Faktoren wie die lange Migrationsdauer und die damit verbundenen Kosten aus.

Migrationsfabrik

Migrationsfabriken setzen hauptsächlich auf die automatisierte Migration von Altsystemen. Es wurde festgestellt, dass ca. 70 bis 75 % der Migrationsaufgaben sich stets wiederholen und daher automatisiert werden können. Migrationsaufgaben, die sich nicht automatisieren lassen, können ohne weiteres an externe Dienstleister ausgelagert werden. Hierbei spielt die Qualitätssicherung eine entscheidende Rolle.

Bei der Verwendung einer *Migrationsfabrik* wird in den folgenden Schritten vorgegangen:

- Projektvorbereitung
- Analyse
- Umstellungsvorbereitung
- Umstellungsdurchführung
- Abschlussarbeiten

Was die eigentliche Umstellung betrifft, kennt man bei der *Migrationsfabrik* drei unterschiedliche Vorgehensweisen. Dies sind die *Punktumstellung* („Big-Bang“), die *Langfristumstellung* und die *Paketumstellung.* In der Umstellungsphase kann dann auch eine eigene *Migrationsfabrik* entwickelt oder auf bestehende Werkzeuge zurückgegriffen werden.

Da bei der *Migrationsfabrik* sämtliche Umstellungsstrategien angewandt werden können, bietet sie hier ausreichend Flexibilität. Durch den hohen Grad an Automatisierung stellt die *Migrationsfabrik* eine sehr zuverlässige Methode dar, die zudem ökonomisch orientiert ist.

4.1.6 Umstellungs- und Übergabestrategien

Punktumstellung (Big-Bang)

Bei der *Punktumstellung* (vgl. Abbildung 4 unten), wird die Migration im Hintergrund vorbereitet und zu einem festgelegten Termin von einem Tag auf den anderen das alte System auf das neue umgeschaltet. Diese Methode geht mit einem sehr hohen Risiko einher, da schnell etwas schief gehen kann, was zu einem geschäftskritischen Stillstand führen kann. Vorteilig wirkt sich die geringe Zeit aus, die für eine solche Umstellung benötigt wird.

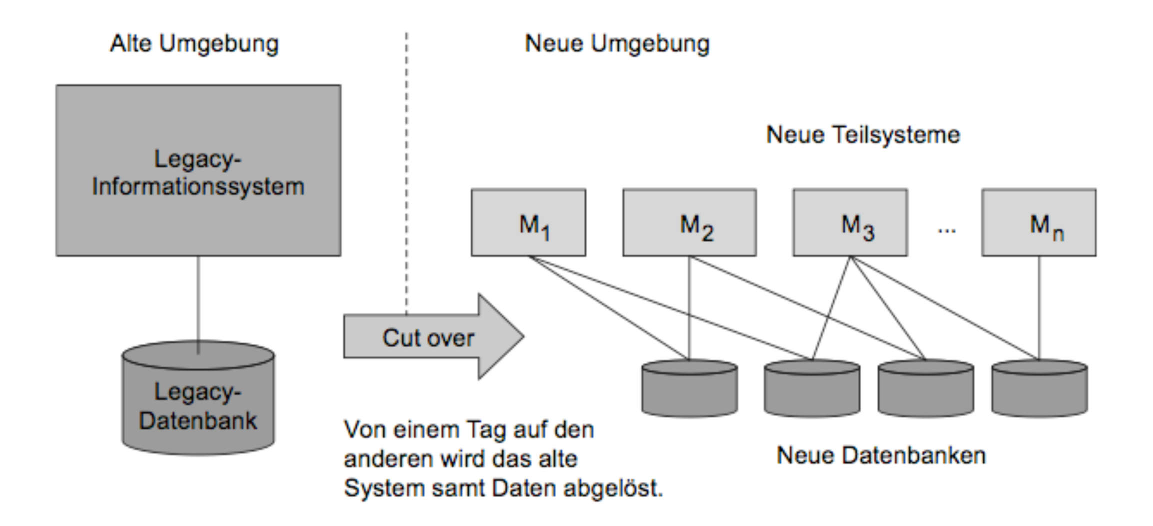


Abb. 4: Punktumstellung[81]

Langfristumstellung

Bei der *Langfristumstellung* werden Teile des Systems genau dann angepasst und migriert, wenn sie ohnehin im Zuge von Wartungsarbeiten angefasst hätten werden müssen. Das Risiko bei dieser Umstellungsstrategie ist recht gering, dennoch dauert dieses evolutionäre Vorgehen enorm lange, was sich wiederum kostensteigernd auswirkt.

[81] Sneed, H/Wolf, E./Heilmann H. (2010), S. 15

Paketumstellung

Wird ein System in mehrere kleine Pakete unterteilt, welche nach und nach migriert werden können, so spricht man von einer *Paketumstellung*. Bei dieser sanften Migration liegt ebenfalls nur ein geringes Risiko vor, da immer nur kleine Stücke des Gesamten angefasst und ausgetauscht werden.

Ausprägungsformen der *Paketumstellung* sind die Paketumstellung mit *Komponentenersatz* und jene mit *Paralleleinsatz*. Bei der Variante mit *Komponentenersatz* werden einzelne Komponenten nach und nach aus dem Legacy-System gezogen und in das neue System migriert. Über Gateways können beide Systeme weiterhin kommunizieren und die einzelnen Komponenten sind voll funktionsfähig. Das System ist in diesem Stadium ein Zusammenspiel von Legacy- und Neu-System, wie in Abbildung 5 visualisiert. Beim *Paralleleinsatz* wird parallel zum Legacy-System ein System mit sämtlichen bereits migrierten Komponenten aufgesetzt. Die Umstellung erfolgt erst nach vollständiger Migration. Bis dahin ist ausschließlich das Legacy-System in Betrieb (Abbildung 6).

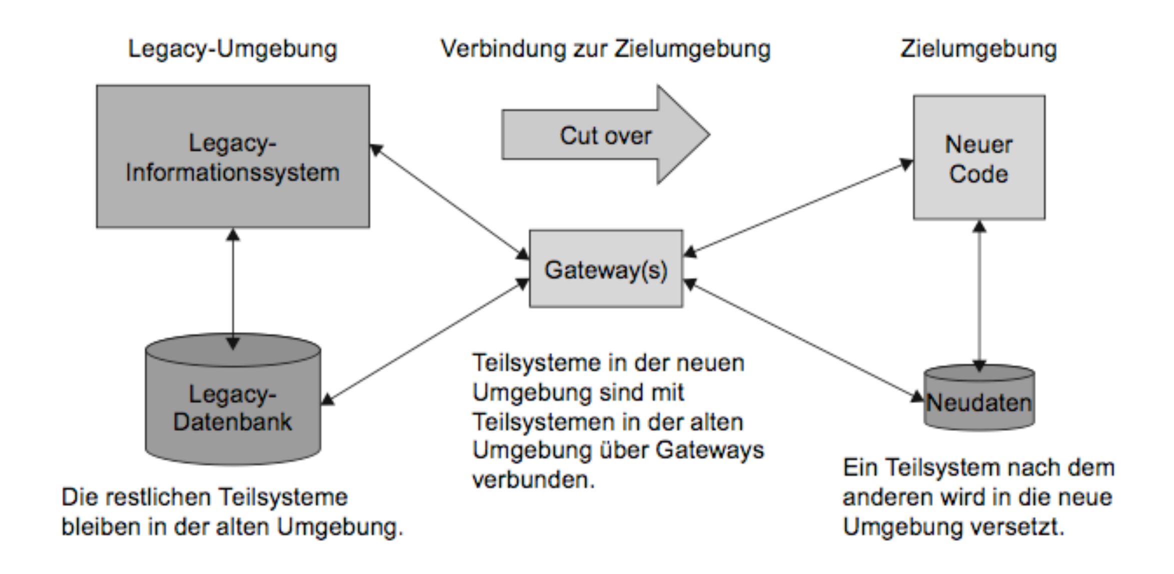


Abb. 5: Paketumstellung mit Komponentenersatz[82]

[82] Sneed, H/Wolf, E./Heilmann H. (2010), S. 17

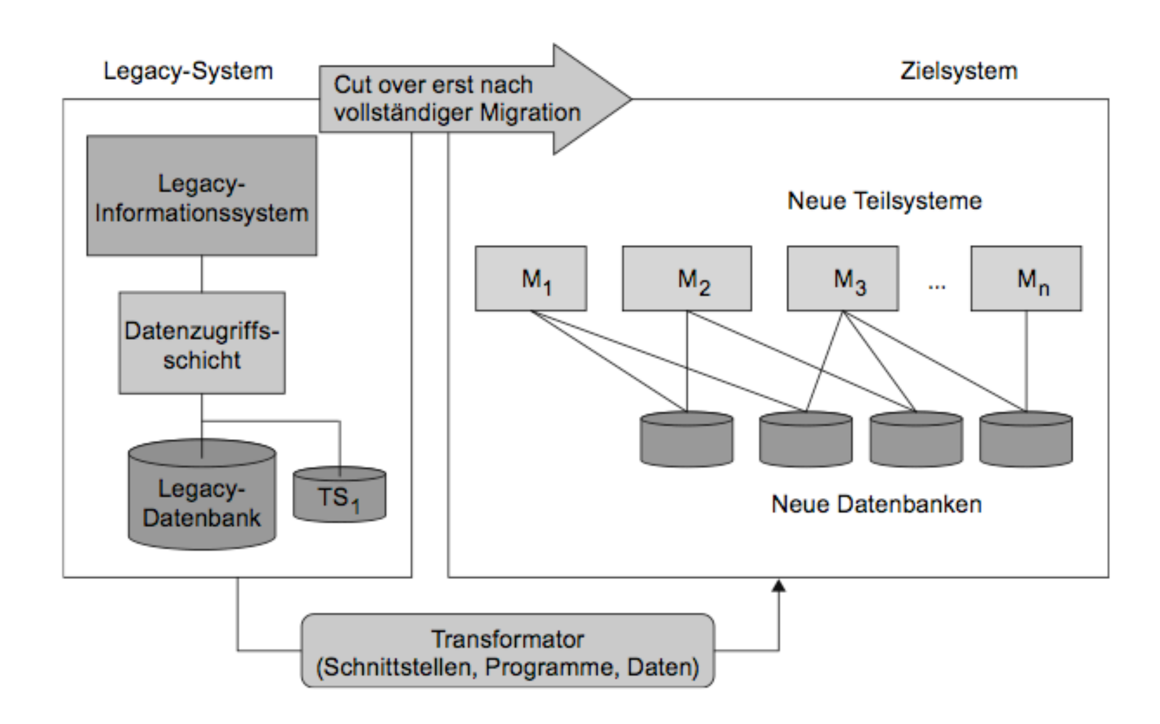


Abb. 6: Paketumstellung mit Paralleleinsatz[83]

4.1.7 Besonderheiten bei Open Source Software

Das Kapitel hat bisher hauptsächlich den Fokus auf allgemeine Grundlagen gelegt und sollte somit einen neutralen Überblick gewähren. In diesem Abschnitt werden nun noch einige Besonderheiten von Migrationen und Open Source Software beleuchtet.

Der Wahrscheinlich größte Vorteil bei einer Migration hin zu OSS ist, dass auf einen großen frei zugänglichen Erfahrungsschatz in der Community oder an anderer Stelle im Internet zurückgegriffen werden kann. Außerdem bringen zahlreiche Dienstleister Knowhow zu diesem Thema mit. Die in den meisten Fällen gute Dokumentation und aktiven Foren können den Migrationsprozess erheblich erleichtern und damit beschleunigen sowie Risiken minimieren. Besonders die zukünftige Nutzung offener Dateiformaten wird es im Endbenutzerbereich erleichtern, geeignete Konverter zu finden, welche die bisherigen proprietären Dateienformate wandeln.

[83] Sneed, H/Wolf, E./Heilmann H. (2010), S. 19

Generell kann jedoch gesagt werden, dass die Migration und deren Ablauf annähernd gleich sind, egal ob es sich um eine Migration von CSS hin zu OSS oder von z.B. CSS zu CSS handelt. Es müssen immer die gleichen oder zumindest ähnliche Schritte durchlaufen werden.

Einige gelungene Beispiele von Migrationen von CSS zu OSS, sowie die Vorgehensweise dabei finden sich in den gelisteten Beispielen am Ende. Durch diese Beispiele kann die theoretische Hülle von Grundlagen dieses Kapitel mit Leben gefüllt werden und das Vorgehen in der Praxis illustrieren.

4.1.8 Checkliste

Checkliste anhand des Beispiels Migration und Umstellung von einem proprietären Groupware-System auf eine Open-Source-Lösung

Betrachtet wird an dieser Stelle abermals das Beispielunternehmen, welches in diesem Leitfaden durchgehend referenziert wird. Zur Bearbeitung des Beispiels wird eine empfehlenswerte (nach Bedarf frei modifizierbare) Checkliste zur Migration auf OSS Schritt für Schritt durchlaufen:

1. Wurde das System ausreichend analysiert?

In diesem Schritt sollte sichergestellt werden, dass das zu migrierende System ausreichen analysiert wurde. Unter analysieren werden hier die u.a. folgenden Punkte verstanden:

- Messung unterschiedlicher Systemkennzahlen wie Größe, Qualität und Komplexität
- Verständnis der IST-Architektur (ggf. Reengineering)
- Verständnis aller Abhängigkeiten

2. Werde das System bewertet?

In diesem Schritt gilt es, das bestehende System anhand der im vorhergehenden Schritt ermittelten Kennzahlen zu bewerten. Hierzu wird das Bewertungsverfahren aus Kapitel *4.1.4 Migrationsvorbereitung* herangezogen.

3. Wurde eine Aufwandsschätzung durchgeführt?

Wurde die Aufwandsschätzung gemäß einer der Methode aus Kapitel *4.1.4 Migrationsvorbereitung* durchgeführt oder eine andere Methode aus der Literatur herangezogen?

4. Ist eine Migration wirtschaftlich sinnvoll?

Nach der Aufwandsschätzung muss die Entscheidung getroffen werden, ob die Migration wirtschaftlich denn auch sinnvoll ist oder ob eine Neuimplementierung evtl. mehr Sinn ergeben würde. Außerdem kann eine Entscheidung über die (teilweise) Abgabe an externe Dienstleister getroffen werden.

Ergebnisse der Wirtschaftlichkeit wurden detailliert bereits in der TCO-Analyse betrachtet. Siehe hierzu Kapitel *3.3 Betrachtung der Total Cost of Ownership*.

5. Ist das System bzw. die Anwendung zerlegbar?

a. Ja, ist zerlgbar (weiter mit Frage 6)

Da es sich hier um eine Groupware-Lösung mit vielen unterschiedlichen Komponenten handelt, welche sowohl Client- als auch Server-seitig in Verwendung sind. Beispiele für die nach und nach migrierbare Komponenten sind Verzeichnis- und Adressdaten (LDAP), E-Mail (Mail-Server) und Kalender. Hinzu können Komponenten wie Datenbanken, Anwendungslogik und auch die Client-Software kommen.

b. Nein, ist nicht zerlegbar (weiter mit Frage 7)

Dieser Punkt trifft auf das Beispiel nicht zu. Meist trifft dieser Punkt auf kleine, monolithische Anwendungen zu oder sehr alte, bei denen noch keine strikte Komponententrennung stattfand und eine starke Verflechtung vorherrscht.

6. Erscheint eine inkrementelle Migration als sinnvoll?

a. Ja

Eine inkrementelle Migration ergibt hier Sinn, da das Risiko eines Systemstillstandes (insbesondere bei migrationsunerfahrenen Unternehmen problematisch) geringer ist.

b. Nein

7. Welche Migrationsstrategie soll verfolgt werden?

Da es sich hier um ein mittelständisches Unternehmen handelt und größtenteils Standardsoftware eingesetzt wird, erscheint ein hoch automatisierter Ansatz am sinnvollsten. Es wird deshalb der Ansatz der Migrationsfabrik gewählt, bei dem ein großer Teil automatisch ablau-

fen kann und nur noch ein kleiner Teil zur händischen Umsetzung bleibt. Dadurch können die absoluten Kosten (die für ein Unternehmen dieser Größe schneller problematisch werden) niedrig gehalten werden.

8. Soll eine Punkt-Umstellung (Big-Bang) erfolgen?

a. Ja (weiter mit Frage 10)

b. Nein (weiter mit Frage 9)

Nein, zu riskant.

9. Welche inkrementelle Umstellung soll erfolgen?

a. Langfristumstellung

Dauert für ein Anwendung dieser Art und Größe zu lange und kostet dementsprechend zu viel.

b. Inkrementelle Paketumstellung mit Komponentenersatz

Ist mit zu viel Aufwand verbunden und bietet für unsere Organisation nicht den entsprechenden Mehrwehrt.

c. Inkrementelle Paketumstellung mit Paralleleinsatz

Es erscheint am sinnvollsten, die inkrementelle Paketumstellung mit Paralleleinsatz zu verwenden, da hier ohne großes Risiko und ohne großes Vorwissen eine ordentliche Migration möglich ist. Das neue System kann schrittweise aufgebaut werden, während das alte bis zum Schluss unangetastet seinen Dienst verrichtet.

10. Sollen Angebote von externen Dienstleistern eingeholt werden?

Es ist ratsam, bei einer Migration, insbesondere eines geschäftskritischen Produktivsystem einen externen Dienstleister mit an Bord zu holen, welcher sich mit der Materie der Migration wesentlich besser auskennt und die hausinterne IT unterstützen kann.

11. Wurden die Anwender vor Beginn der Migrationsarbeiten informiert?

Die Mitarbeiter wurden zum einen darüber informiert, dass eine Migration des bestehenden Nachrichten- und Groupware-Systems auf eine OSS-Lösung stattfinden wird. Sie wurden auch über die der Migration zu Grunde liegende Motivation des Managements informiert. Es

ist vorgesehen, die Anwender im Rahmen der anstehenden Software-Schulungen über Vorteile der Migration zu informieren, die sie unmittelbar tangieren. Beispiele hierfür sind bessere Bedienbarkeit und erhöhte Performance. Die späteren Anwender werden regelmäßig über den Fortschritt der Migration in Kenntnis gesetzt, um sie aktiv miteinzubinden und dadurch die Akzeptanz der OSS-Lösung zu steigern.

12. Wurden ausreichen Rollback-Maßnahmen getroffen?

Da sich für eine inkrementelle Paketumstellung mit Paralleleinsatz entschieden wurde, sind eigentlich keine Rollback-Maßnahmen nötig. Die Anwender arbeiten bis zur Umstellung ohnehin weiter auf dem Legacy-System. Dennoch ist es prinzipiell sinnvoll, bei der Migration Sicherungspunkte zu erstellen, um Fehler schneller wieder rückgängig machen zu können.

13. Wurde das System nach erfolgreicher Migration getestet?

Nach der Migration wurde das System erfolgreich getestet und kann freigegeben werden. Nach der Umstellung werden alle Anwender auf der neuen Umgebung produktiv arbeiten können.

Literaturempfehlung:

Sneed, H/Wolf, E./Heilmann H. (2010): Softwaremigration in der Praxis, Übertragung alter Softwaresysteme in eine moderne Umgebung

Bundesministerium des Inneren (2008): Migrationsleitfaden, Leitfaden für die Migration von Software, URL: http://www.cio.bund.de/DE/Architekturen-und-Standards/Migrationsleitfaden-und-Migrationshilfen/migrationsleitfaden_node.html

4.2 Anwenderschulung und Dokumentation

Nachdem die OSS implementiert wurde, gilt es, die Mitarbeiter in der Anwendung der OSS zu schulen. Dabei kann man grundsätzlich zwischen zwei Möglichkeiten entscheiden. Entweder die Schulung wird eingekauft oder die Schulung wird vom Unternehmen selbst organisiert.

4.2.1 Schulung durch einen externen Anbieter

Der Vorteil des erstgenannten Falls ist, dass die Verantwortung abgegeben wird und keinerlei Schulungsvorbereitungen zu treffen sind. Durch eine Onlinerecherche lassen sich Anbieter von Schulungen für die zu implementierende OSS finden. Dabei empfiehlt es sich, darauf zu achten, dass der ausgewählte Schulungsleiter eine anerkannte Zertifizierung im zu lehrenden Fach aufweisen kann. Nachdem ein passender Schulungsanbieter ausgewählt wurde, gilt es, sich für die passende Schulungsart zu entscheiden. Häufig bieten die Schulungsleiter eine Auswahl von mehreren Schulungsarten an, zwischen denen es sich zu entscheiden gilt. Im Folgenden werden die häufigsten Formen aufgeführt. Es werden die Schulungsarten auf ihre Vor- und Nachteile hin analysiert und potenzielle Einsatzgebiete vorgeschlagen, um eine Hilfestellung bei der Auswahl der richtigen Schulung an die Hand zu geben[84]:

Schulungsart	Kurzbeschreibung	Vorteile und Nachteile	Einsatzgebiete
Frontalschulung	Klassische Form der Schulung, wobei der Schulungsleiter die Inhalte des Kurses lehrt	Vorteile: - Kostengünstig - Methodenkompetenz des Schulungsleiters Nachteile: - Begrenzte Aufnahmefähigkeit der Schulungsteilnehmer aufgrund von unilateralem Vortrag - Keine Möglichkeit, den Wissenstand aller Kursteilnehmer zu erfassen und den Kursinhalt auf alle Kursteilnehmer anzupassen - Fehlende Flexibilität	Geringer Komplexitätsgrad der Software; zur Vermittlung von Grundkenntnissen geeignet
Individual-schulung	Gruppengröße ist auf drei Gruppenmitglieder begrenzt	Vorteile: - Individuelle Ansprache der Lernbedürfnisse - Höhe Effizienz - Hohe Flexibilität bei der	Bei hohem Komplexitätsgrad der Software besonders geeignet.

[84] Vgl. Computerschule (o.J.)

		Terminwahl Nachteile: - Höhere Kosten - Geringeres Erfahrungsspektrum durch wenige Gruppenmitglieder	
Gruppenschulung/Workshop	Themen werden aktiv und teilweise kreativ in Gruppen erarbeitet	Vorteile: - Großes Erfahrungsspektrum durch viele Gruppenmitglieder - Interaktives Lernen - Durch sinnvolle gruppendynamische Konstellationen kann gegenseitig Wissen vermittelt werden. Nachteile: - Fehlende Motivation bei Gruppenmitgliedern kann die Motivation der gesamten Gruppe kippen - Fehlende Flexibilität	Bei hohem Komplexitätsgrad und nötigem Verständnis der Zusammenhänge einsetzbar
E-Learning	Lernen mit elektronischer Unterstützung	Vorteile: - Flexibel anwendbar - Problemorientiert - Individuelle Kenntnisermittlung und Weiterbildung - Geringe Teilnehmerkosten - Kein Zeitdruck, jeder Mitarbeiter kann sein eigenes Lerntempo festlegen - Multimediatechnologie Nachteil: - Hohe Anschaffungskosten - Neue Inhalte brauchen lange, bis sie vermittelt werden können	Bei sehr großer Anzahl von Schulungsteilnehmern, geringem Komplexitätsgrad und zur allgemeinen Wissensvermittlung

Tabelle 6: Schulungsarten

Welche Schulungsart die passende ist, gilt es individuell nach den Vorstellungen, Voraussetzungen und Zielsetzungen jeder Organisation zu entscheiden.

4.2.2 Schulung durch interne Mitarbeiter

Als zweite Möglichkeit steht die interne Durchführung der Schulung zur Auswahl. Dies kann entweder passiv durch das Bereitstellen von Schulungsunterlagen oder aktiv durch einen Mitarbeiter des eigenen Unternehmens vollzogen werden.

Passive Schulung

Bei der passiven Schulung der Mitarbeiter wird auf die Eigenverantwortung und Motivation der Mitarbeiter gesetzt. Das Unternehmen hat die Aufgabe den Mitarbeitern Zeit und Schulungsdokumente zur Verfügung zu stellen, damit diese sich das Knowhow in Eigenverantwortung aneignen können.

Bei dieser Schulungsmethode kann mit wenig Kosten und geringem Zeitaufwand gerechnet werden. Jedoch ist die Effektivität der Schulung geringer, da damit gerechnet werden muss, dass nicht jeder Mitarbeiter ausreichend motiviert sein wird, um sich selbst die nötigen Kenntnisse anzueignen. Für Rückfragen empfiehlt es sich, einen oder mehrere OSS-Experten (so genannte *Champions*) festzulegen, die für alle Mitarbeiter transparent gemacht werden. Sollte man sich für diese Schulungsmethode entscheiden, empfiehlt sich folgendes Vorgehen:

Als erstes sollte das passende Schulungsmaterial ausgewählt bzw. zusammengestellt werden. Der Vorteil bei Schulungen zu OSS ist, dass häufig Dokumentationen oder sogar Onlinetrainings kostenlos zur Verfügung stehen. Diese sind in der entsprechenden Open Source-Community und durch eine Onlinerecherche zu finden. (Beispielsweise wird auf der Eclipse-Homepage eine Dokumentation zu Eclipse kostenlos zur Verfügung gestellt[85]). Diese Schulungsunterlagen oder selbst konzipierte Unterlagen werden anschließend den Mitarbeitern zur Verfügung gestellt. Es gilt einen Zeitraum festzulegen, in dem sich die Mitarbeiter das nötige Wissen aneignen. Es muss den Mitarbeiter deutlich gemacht werden, dass das Wissen bis zu dieser Frist vorhanden sein muss, damit das Tagesgeschäft aufgrund von fehlendem

[85] Zu finden unter folgendem Link: http://www.eclipse.org/documentation/

Knowhow nicht negativ beeinflusst wird. Eine Möglichkeit, die Motivation der Mitarbeiter zu erhöhen ist es, das Erlernen der OSS als (gehaltsbestimmendes) Ziel in die Jahreszielvereinbarung aufzunehmen, falls das Unternehmen solche Vereinbarungen pflegt. Das Wissen kann durch einen Test abgefragt werden, damit sichergegangen werden kann, dass die Mitarbeiter den Umgang mit der Software beherrschen.

Aktive Schulung

Soll eine aktive Schulung stattfinden, müssen entweder interne Mitarbeiter gefunden werden, die im Umgang mit der Software vertraut sind und ihr Knowhow an ihre Kollegen weitergeben können oder zukünftige *Key-User* durch einen externen Anbieter ausgebildet werden, die anschließend die Kollegen schulen können. Als *Key-User* eignen sich besonders Mitarbeiter, die bereits einen Expertenstatus für die Altanwendung innehaben. Experten tendieren dazu, der Migration auf eine neue Software negativ gegenüber zu stehen, da sie befürchten ihren Expertenstatus zu verlieren. Deshalb ist empfehlenswert diese Experten als *Key-User* einzusetzen, damit sie ihren Expertenstatus auch für die neue Software behalten, der Migration gegenüber positiv gestimmt sind und diese unterstützen.

Nachdem diese Entscheidung getroffen wurde, werden Lernziele definiert, um die Schulung entsprechend konzipieren, aufbauen und den Lernerfolg messen zu können. Außerdem muss eine Schulungsart (siehe Tabelle 6) ausgewählt werden, deren Durchführung geplant werden muss. Aspekte dieser Planung sind die Erstellung eines Ablaufplans (Welche Inhalte werden wann vermittelt?) sowie die Auswahl und Planung der Lehrmethode.

4.2.3 Kontrolle und Feedback

Das Modell von Donald Kirkpatrick liefert einen ganzheitlichen Ansatz zur Evaluierung des Erfolgs einer Schulung. Das Modell, das im Original „Evaluating Training Programs: The Four Levels“ [86] genannt wird, ist in vier Ebenen aufgeteilt.

[86] Kirkpatrick, D./Kirkpatrick, L. (2008), S. xi

Erste Ebene („Reaction“) [87]

Hier ist das Ziel, die Zufriedenheit der geschulten Mitarbeiter zu messen, um deren konstruktive Rückmeldung zur Verbesserung der Schulung zu nutzen. Es empfiehlt sich, einen Feedbackbogen zu erstellen, der an alle Schulungsteilnehmer am Ende der Schulung ausgeteilt wird. Beim Erstellen des Bogens muss überlegt werden, welche Informationen damit in Erfahrung gebracht werden sollen. Im Folgenden werden einige Fragen zur Orientierung gelistet, die in einen Feedbackbogen aufgenommen werden können:

1. Wie zufrieden waren Sie mit der Schulung/dem Schulungsleiter/dem Schulungsmaterial?
2. Fühlen Sie sich im Umgang mit der neuen OSS sicher?
3. Was kann Ihrer Meinung nach verbessert werden?

Zweite Ebene („Learning“) [88]

Diese Ebene zielt auf die Überprüfung des Gelernten ab. Hierbei bietet sich ein theoretischer Test zur Ermittlung des Hintergrundwissens über die OSS Software an und zusätzlich ein praktischer Test, der zeigen soll, wie viel die Mitarbeiter im Umgang mit der OSS Software gelernt haben. Es gilt herauszufinden, ob und wie gut die Mitarbeiter mit der neu implementierten Software zurechtkommen.

Dritte Ebene („Behavior“) [89]

Diese Ebene soll nach Kirkpatrick überprüfen, wie sehr sich der Berufsalltag der geschulten Mitarbeiter durch das Training verändert hat. Zu überprüfen gilt es, in wie weit sich das Verhalten der Mitarbeiter – möglichst objektiv bewertet – gewandelt hat. Dabei ist wichtig zu evaluieren, wie die neu eingesetzte OSS Software Tagesabläufe und Prozesse verändert hat.

---

[87] Vgl. Kirkpatrick, D./Kirkpatrick, L. (2008), S.40 ff.
[88] Vgl. ebd., S.63 ff.
[89] Vgl. ebd., S.78 ff.

Vierte Ebene („Result“) [90]

Diese Ebene dient dazu, Betriebskennziffern zu überprüfen. Es gilt also festzustellen, wie sich beispielsweise der Gewinn der Unternehmung, die benötigte Zeit für eine definierte Arbeit, die Arbeitsqualität und die Kosten verändert haben. Dies ist auf Basis der in der dritten Ebene festgestellten Veränderungen durchzuführen.

4.2.4 Dokumentation

Alle Erkenntnisse und das Wissen, das bei der Vorbereitung und Implementierung der OSS gewonnen wurde, sollten dokumentiert werden.
Die Dokumentation sollte intern von einem oder auch mehreren Mitarbeiter übernommen werden, die den größten Gesamtüberblick über die Abläufe bei der Einführung von OSS haben. Zu dokumentieren sind u.a. folgende Informationen:

- Gründe für die Migration
- Entscheidungskriterien, nach denen die OSS ausgewählt wurde
- Handhabung der OSS
- Best Practices
- Probleme und wie sie gelöst wurden
- Zugriffsmöglichkeit auf Schulungsmaterial
- Ansprechpartner und deren Zuständigkeitsbereiche

Die Dokumentation sollte allen Mitarbeitern frei zur Verfügung gestellt werden, damit diese nach der Schulung zur Weiterbildung, zum Ausbau und zur Festigung des Wissens und als Nachschlagewerk verwendet werden kann. Um den Zugriff zu ermöglichen, wäre eine Bereitstellung der Dokumentation im Internet, Intranet oder als Software auf den Workstations denkbar.

[90] Vgl. Kirkpatrick, D./Kirkpatrick, L. (2008), S.96 ff.

4.2.5 Besonderheiten bei OSS als Schulungsgegenstand

In vielen Fällen stehen Mitarbeiter der Einführung einer OSS skeptisch gegenüber, da sie befürchten, dass das Unternehmen hiermit nur Kosten einsparen möchten und die neu implementierte OSS in der Anwendung Schwierigkeiten bereiten wird.

Um den Mitarbeitern diese Unsicherheit zu nehmen, empfiehlt es sich, während der Mitarbeiterschulung auch einen Fokus auf die Vorteile der Migration der OSS zu legen. Es gilt, den zusätzlichen Nutzen und hilfreiche Funktionen, die durch den Umstieg von einer proprietären Software auf eine OSS entstehen, besonders hervorzuheben, um die Mitarbeiter davon zu überzeugen, dass die neue Softwarelösung tatsächlich Vorteile (und nicht nur Kosteneinsparungen) mit sich bringt.

4.2.6 Checkliste

Schulungsauswahl und -durchführung

1. Soll die Schulung durch einen externen Anbieter (weiter mit 2.) durchgeführt oder von internen Mitarbeitern (weiter mit 4.) übernommen werden?

Die Schulung wird von einem externen Anbieter durchgeführt, da intern das technische Wissen nicht komplett vorhanden ist, und keine didaktische Erfahrung vorliegt.

2. Welcher externe Anbieter wird eingesetzt?

Ein Softwareunternehmen, das als Partner für die Open-Source-Lösung agiert und Erfahrung mit verschiedenen Schulungsarten hat, wurde ausgewählt.

3. Welche Schulungsart soll angewendet werden?

Es wird ein E-Learning genutzt. Dieses bietet für das Beispielunternehmen zwei Vorteile.
Die Mitarbeiter des Unternehmens sind in der Dienstleistungsbranche beschäftigt (und nicht am Fließband) und treten primär per E-Mail und Telefon mit Kunden in Kontakt (und nicht persönlichen in einer Filiale). Dabei entstehen gelegentlich Zeiten, in denen Mitarbeiter nicht beschäftigt sind. Das E-Learning kann während dieser Zeiten durchgeführt werden, wodurch die Schulungsmaßnahme keine zusätzliche Arbeitszeit in Anspruch nimmt. 250 Mitarbeiter zu schulen, würde ansonsten Zeit- und Kostenprobleme im Unternehmen verursachen.

Außerdem kann das E-Learning von den Mitarbeitern später als Nachschlagewerk genutzt werden, welches sie im Vergleich zu einem Handbuch (das oft eher ungelesen bliebe) bereits kennen. Die Erstellung des E-Learnings wird an einen externen Spezialisten für die Software mit Schulungserfahrung ausgelagert. Die dadurch entstehenden Kosten werden in Kauf genommen, da intern nicht ausreichend Wissen bezüglich Gestaltung und Didaktik vorhanden ist. Das E-Learning wird komplett zum Eigentum des Unternehmens.

4. Entscheidung für interne Mitarbeiter: Aktive (weiter mit 5.) oder passive Durchführung (weiter mit 8.) der Schulung?

Nicht zutreffend.

5. Aktive Schulung:

Sollen Key-User durch externe Anbieter geschult oder Mitarbeiter mit Vorwissen herangezogen werden?

Nicht zutreffend.

6. Key User:

Nicht zutreffend.

7. Mitarbeiter mit Vorwissen:

Nicht zutreffend.

8. Passive Schulung:

Selbstkonzipierte Schulungsunterlagen (weiter mit 9.) oder kostenlose, online verfügbare Dokumentationen (weiter mit 10.) zur passiven Schulung anbieten?

9. Selbstkonzipierte Schulungsunterlagen

Nicht zutreffend.

10. Kostenlose, online verfügbare Dokumente

Nicht zutreffend.

11. Evaluation und Feedback

Wurden alle Ebenen von Kirckpatrick geprüft?

12. Reaction:

Die Umfrage hat ergeben, dass die Mitarbeiter mit der Schulung zufrieden waren.

13. Learning:

Die Mitarbeiter haben sowohl im theoretischen, als auch im praktischen Test gut bis sehr gut abgeschnitten. Dies, obwohl der Test auch anspruchsvolle Konfigurationsmöglichkeiten abgefragt hat.

14. Behavior:

Die Mitarbeiter kommen gut mit der OSS Lösung zurecht. Einige Arbeitsschritte empfinden sie sogar als einfacher und unkomplizierter als mit der alten Lösung.

15. Result:

Die Mitarbeiter können mit der OSS effizient arbeiten.

Literaturempfehlung:

D. Kirkpatrick/ L. Kirkpatrick (2008): Evaluating Training Programs, The Four Level, 3. Auflage, San Francisco: Berrett-Koehler Publishers

Planung, Vorbereitung, Durchführung und Erfolgskontrolle einer Schulung.

5 Produktivphase

Nachdem die ausgewählte Software in der Transformationsphase erfolgreich implementiert wurde und die Anwender das nötige Wissen zur Arbeit mit der neuen Lösung vermittelt bekommen haben, gilt es in der Produktivphase, sicherzustellen, dass die Software kontinuierlich produktiv eingesetzt werden kann. Dies kann vor allem durch Support und Wartung gewährleistet werden. Diese beiden Unterstützungsprozesse können prinzipiell (kostenpflichtig) von einem externen Dienstleister oder (kostenlos bzw. kostengünstig) von der hinter der OSS stehenden Community bezogen werden.

5.1 Interaktion mit der Community

Entschließt sich eine Organisation dazu, für einen bestimmten Zweck OSS (an Stelle von CSS) zu verwenden, ergeben sich hieraus auch Konsequenzen für die Kommunikation mit den Entwicklern der Software. In aller Regel wird nicht - wie bei einer proprietären Lösung - ein vertraglich gebundener Lieferant als Ansprechpartner bei Fragen, Problemen oder Änderungswünschen bezüglich der Software zur Verfügung stehen. Kauft die Organisation diese Beratungsdienstleistungen nicht von einem hierauf spezialisierten Drittanbieter ein, ist sie auf eine zielführende Kommunikation mit der zur jeweiligen OSS gehörigen Entwickler-Community angewiesen. Hierzu hilft es auch, die Motivationsstruktur von Entwicklern in OSS-Projekten zu verstehen.

Dieses Kapitel soll dazu anregen, einige Gedanken auf die Frage nach einer angemessenen Kommunikationsstrategie mit der hinter der einzuführenden OSS stehenden Community zu verwenden. Es werden einige grundlegende Fragestellungen aufgeworfen und zu diesen jeweils empfehlenswerte Vorgehensweisen dargestellt.

5.1.1 Zeitpunkt der Interaktion

Eine Frage, die man sich im Interesse der eigenen Organisation möglichst früh stellen sollte, ist die, ab welchem Zeitpunkt man mit der Community in Kontakt tritt. Prinzipiell denkbar wäre es, erst beim Auftreten des ersten Problems oder der ersten Frage Initial mit diesem konkreten Anliegen an die Community heranzutreten. Eine vorausschauendere und chancenreiche Alternative läge jedoch darin, so früh wie nur möglich einen geregelten Kontakt zu

etablieren. Den Zeitpunkt der Kontaktaufnahme mit der Community nach vorne zu verlegen, bringt keine ersichtlichen Nachteile mit sich. Vielmehr kann hierdurch möglicher Weise bereits vor dem konkreten Rollout der Software in der Organisation die weitere Entwicklung des Produkts in die gewünschte Richtung hin beeinflusst werden. Auch kann mit Hilfe bereits vorab implementierter Kommunikationswege durch gezieltes Adressieren der Anfrage beim Auftreten eines ersten Unterstützungsbedarfes schneller reagiert werden. Selbst wenn eine frühzeitige Kommunikation seitens der eigenen Organisation nur passiv erfolgt (z.B. durch das Lesen von Wikis und Foreneinträgen), lässt sich hiermit bereits vor dem Rollout der OSS intern grundlegendes Knowhow anreichern.

5.1.2 Zweck der Interaktion

Eine weitere Grundsatzfrage, die sich die eigene Organisation stellen sollte, ist die nach dem Zweck der Kommunikation mit der Entwickler-Community: Soll nur dann kommuniziert werden, wenn die Organisation Unterstützung benötigt (Rolle: Leser) oder soll eine partnerschaftliche Beziehung avisiert werden, in der es zu einer wechselseitigen Unterstützung und regelmäßiger Kommunikation kommt?

Für den zuerst genannten Ansatz spricht die Tatsache, dass Kommunikation immer auch Zeit (und damit in den allermeisten Fällen Geld) in Anspruch nimmt. Die Motivation, Aufwände für die Kommunikation mit der Community gering zu halten, leuchtet zunächst ein. Allerdings sollte man stets bedenken, dass das Antwortverhalten der Community möglicher Weise stark davon abhängt, in welchem Maße man sich in der Vergangenheit selbst beteiligt hat. Wenn die Organisation dazu beiträgt, Tutorials zu verfassen, Bugs zu identifizieren (Rolle: *Bug Reporter*) sowie Lösungen bereitzustellen (Rolle: *Bug Fixer*) und diese Mithilfe transparent machen kann, wird dies vermutlich positive Auswirkungen auf die Hilfsbereitschaft der Community haben (wichtige Motivationsfaktoren für viele OSS-Entwickler sind die *Identifikation mit der Gruppe* sowie ein gewisser *Altruismus*, der dem Prinzip von gerechtem Geben und Nehmen folgt[91]). Sollte die Organisation die verwendete OSS für betriebliche Zwecke um bestimmte Funktionen erweitern und werden diese nicht als geschäftskritisch angesehen, so besteht darüber hinaus die Möglichkeit, diesen eigens entwickelten Code der Community

[91] Vgl. Luthiger, B. (2004), S. 4

bereitzustellen (Rolle: *Gelegenheits-Entwickler*). Auf diese Weise kann die gesamte Nutzerschaft von der Entwicklung profitieren und gleichzeitig besteht die Möglichkeit, dass dieser Code in einem späteren Release eingebunden und dadurch gepflegt und weiterentwickelt wird. Eine weitere Möglichkeit, der Community einen Mehrwert bestünde darin, sehr gutes Schulungsmaterial aus der Organisation allen Nutzern bereitzustellen. Im Falle des Referenzunternehmens aus dem Ratgeber würde es sich beispielsweise lohnen, das eigens in Auftrag gegebene E-Learning (das keine unternehmensspezifischen Inhalte enthält) mit der Community zu teilen. Natürlich kann neben einem Beitrag auf den bisher genannten Wegen auch eine finanzielle Unterstützung erfolgen. Wie bereits in Kapitel *3.2.2 Bewertung der Software* erwähnt, sind in etwa 42% der beteiligten Entwickler an einem Open Source-Projekt bezahlt und angestellt bei einem Unternehmen. Open Source-Projekte haben dennoch einen gewissen Finanzierungsbedarf, der durch finanzielle Zuwendungen von Nutzern zu einem gewissen Teil gedeckt werden kann. Durch eine Geldspende wird also auch dazu beigetragen, das Projekt fortzuführen sowie Support und Anpassungsarbeiten aufrecht zu erhalten.

Durch die aktive Mitarbeit in der Community lässt sich zudem eine positive Innenwirkung erzeugen: Die Mitgestaltung folgt der Ideologie des Open Source-Gedankens und kann bei den jeweiligen Akteuren beispielsweise das Bedürfnis nach Selbstverwirklichung befriedigen. Auch der Lerneffekt, der beim Lösen von Problemen mit der OSS erzielt wird, sollte als Humankapital für die eigene Organisation nicht vernachlässigt werden.

5.1.3 Definition der Kommunikationswege

Besondere Bedeutung sollte auch der Gestaltung der Kommunikationskanäle beigemessen werden. Diese resultiert aus der Beantwortung zweier hauptsächlicher Fragestellungen:

- Wer kommuniziert im Namen der eigenen Organisation mit der Community?
- Mit wem wird auf Seiten der Community überhaupt Kontakt aufgenommen?

In Bezug auf die erste Frage empfiehlt der im Kontext des EU-Projektes „FLOSSMetrics“ entstandene “SME guide to Open Source Software”, klare Verantwortliche innerhalb der Organisation zu bestimmen, die geregelt mit der Community kommunizieren. Diese Verantwortlichen sollten sich kontinuierlich passiv Wissen aneignen (Verfolgung von Blogs, Wikis, Foren), aktiv Nachfragen an die Community richten und nach Möglichkeit in der Community

ihrerseits (Wissens-) Beiträge leisten. Die Community-Beauftragten sollten ein gewisses Gespür dahingehend mitbringen oder vermittelt bekommen, dass die Kommunikation mit freiwilligen Entwicklern sehr viel mehr Feinfühligkeit erfordert als die mit einem Vertragspartner. Letzteren kann man in letzter Konsequenz immer auf seine vertragliche Verpflichtung hinweisen. Die Mitglieder einer Community allerdings unterliegen einer solchen Verpflichtung nicht und müssen daher auf andere Weise motiviert werden, im Interesse der Organisation zu handeln. Eine weitere wichtige Aufgabe dieser "OSS-Botschafter" sollte es sein, der restlichen Organisation den Zugang zu relevantem Wissen zu erleichtern. Dies kann beispielsweise in Form einer Intranetseite geschehen, auf der die Lösungen von organisationsspezifischen Problemen dokumentiert werden, aber auch sinnvoll kategorisiert auf weiterführende externe Wissensquellen verlinkt wird.[92]

Bezüglich der Frage danach, wer auf Seiten der Community kontaktiert werden sollte, lassen sich zwei grundsätzliche Strategien aufzeigen: Einerseits ist es denkbar, über die frei zugänglichen Kommunikationskanäle (v.a. Foren) Kontakt mit der gesamten Community aufzunehmen. Hieraus ergeben sich folgende Vorteile: Der Zeitaufwand und die Kosten, die unmittelbar auf die Kommunikation entfallen, sind tendenziell gering (einen Thread in einem Forum zu eröffnen und zu pflegen kostet in aller Regel lediglich Zeit zum Lesen und Schreiben der Beiträge, welche zudem flexibel eingeteilt werden kann). Außerdem wird mit dieser Herangehensweise potenziell auf das Wissen der gesamten Community zugegriffen. Diese Vorgehensweise setzt an den Motivationsfaktoren *Identifikation mit der Gruppe*, *Lernen*, *Altruismus* und *Spaß* an[93]. Es besteht allerdings die Gefahr, dass die eigenen Anfragen in einer Vielzahl von weiteren Beiträgen verschüttet gehen und sich so die Dauer bis zur Lösung eines Problems enorm in die Länge ziehen kann.

Eine alternative Herangehensweise besteht darin, sich unmittelbar mit den Kernmitgliedern (*Core Members*[94]) oder sogar den Projektleitern (*Project Leaders*[95]) in Verbindung zu setzen (per Mail, IRC oder auf Open-Source-Veranstaltungen) und diese dazu zu motivieren, sich an der Lösung eines Problems zu beteiligen. Der Vorteil hierbei liegt auf der Hand: Gelingt es,

[92] Vgl. Daffara, C. (2009), S. 31
[93] Vgl. Luthiger, B. (2004), S. 4 ff.
[94] Vgl. Ye, Y./Kishida, K. (2003), S.2
[95] Vgl. ebd.

die Hauptverantwortlichen des Projekts im eigenen Interesse zu beeinflussen, steigen die Erfolgsaussichten und die Geschwindigkeit der Problemlösung enorm. Problematisch sind jedoch einerseits die Tatsache, dass man nicht mehr unmittelbar auf das Knowhow der gesamten Community zurückgreift (wobei vermutlich die Hauptverantwortlichen dies mittelbar tun werden) und andererseits die in aller Regel höher ausfallenden Kosten. Die Hauptverantwortlichen eines OSS-Projekts werden sich tendenziell den Anspruch, exklusiv mit ihnen zu kommunizieren und sie für die Problemlösung im Sinne der eigenen Organisation einzusetzen, kompensieren lassen. Denkbare Motivatoren für diese Personengruppe sind zunächst Geldzuwendungen an einzelne Entwickler oder die gesamte Community (z.B. in Form von Wettbewerben oder Sponsoring), aber durchaus auch die Aussicht, mit einem erfolgreichen Projekt hausieren zu können oder einen attraktiven Job innerhalb der Organisation zu ergattern (Motivationsfaktor *Reputation und Signalproduktion*[96]). Darüber hinaus ist zu erwarten, dass sich der Anspruch an die Kommunikationskompetenzen im Dialog mit den Hauptentwicklern (durch die direkte und persönliche Ansprache einer Person) noch einmal erhöhen wird.

Abschließen lässt sich feststellen, dass es sich empfiehlt, bei weniger dringenden und weniger speziellen Problemen, auf konventionelle Art die gesamte Community anzusprechen und im Falle äußerst organisationsspezifischer und/oder zeitkritischer Probleme die Kosten für einen direkten Kontakt zu den Hauptverantwortlichen der Community in Kauf zu nehmen.

5.1.4 Checkliste

1. Ab welchem Zeitpunkt wird mit der Community kommuniziert?

Das Unternehmen beginnt die passive Kommunikation bereits zu Beginn der Evaluationsphase, indem sich die verantwortlichen Mitarbeiter in Wikis und Tutorials (*best practices*) zur Einrichtung der OSS-Groupware einlesen. Außerdem versuchen sie durch Lesen der Community-Foren, bereits vorab häufig auftretende Probleme mit der Migration und dem Betrieb der OSS-Groupware zu identifizieren, um diese früh vermeiden zu können.

[96] Vgl. Luthiger, B. (2004), S.3

Die aktive Kommunikation mit der Community beginnt, sobald konkrete Fragen auftreten, die nicht bereits durch die vorhandene Dokumentation beantwortet werden. Dies kann bereits im Zuge der Evaluationsphase oder aber erst im Verlauf der etwaigen Umstellung von der aktuellen proprietären Lösung auf die OSS geschehen.

2. Zu welchem Zweck wird mit der Community kommuniziert - welche Rolle wird innerhalb der Community eingenommen (Leser, Bug Reporter, Bug Fixer, Gelegenheitsentwickler)?

Das Unternehmen wird zunächst nur mit der Community in Kontakt treten, um Wissen anzusammeln und Fragen zu stellen. Aus seiner heutigen Perspektive ist es nicht nötig, die Community zu Anpassungen an dem OSS Produkt zu veranlassen. Alle funktionalen Anforderungen seitens des Unternehmens werden bereits in der heutigen Form abgedeckt.

Es wird daher zunächst ausschließlich die Rolle des Lesers (und Fragestellers) eingenommen.

Sollten im Verlauf der Arbeit mit dem Produkt Bugs identifiziert werden, werden diese selbstverständlich an die Community weitergemeldet (Rolle: *Bug Reporter*).

Entwicklerisch wird das Unternehmen aller Voraussicht nach nicht tätig werden. Von seinem aktuellen Standpunkt aus muss es keine Anpassungen an der Software vornehmen. Für "ehrenamtliche" Entwicklungsarbeit im Rahmen der Community hat es aktuell keine Kapazitäten verfügbar. Allerdings wird das Unternehmen das bei einem externen Dienstleister in Auftrag gegebene E-Learning über die Benutzung der Groupware der gesamten Community zur Verfügung stellen und so einen Beitrag an der Gemeinschaft leisten.

3. Wer ist seitens des Unternehmens zuständig für die Kommunikation mit der Community?

Bisher hat das Unternehmen einen IT-Mitarbeiter, der zuständig ist für die Administration und den hausinternen Support der proprietären Groupware. Dieser ist maßgeblich zuständig für die Evaluation der OSS und die mögliche Migration. Folglich wird er zukünftig die Rolle des "Community-Botschafters" einnehmen.

4. Welche Personenkreise spricht das Unternehmen innerhalb der Community an (alle vs. Hauptverantwortliche)?

Zunächst wird über die frei verfügbaren Kanäle (Foren, Wikis, Mailinglisten) mit der gesamten Community kommuniziert.

Anlass, im Speziellen auf die Hauptentwickler zuzugehen, wird aktuell nicht gesehen.

5. Werden (monetäre) Mittel zur Unterstützung der Kommunikation mit der Community bereitgestellt (z.B. für Sponsoring)?

Die Hauptmotivation für die Migrationsüberlegungen ist die Kostenersparnis, daher wird zunächst nicht in Betracht gezogen, über die Arbeitskraft des IT-Mitarbeiters hinausgehende Ressourcen in die Kommunikation mit der Community zu investieren.

Literaturempfehlungen

Benno Luthiger: "Alles aus Spaß? Zur Motivation von Open-Source-Entwicklern"

Eine gute Übersicht über die Motivationsstruktur von OSS-Entwicklern und die Möglichkeiten, sich in eine Community zu engagieren.

Yunwen Ye, Kouichi Kishida: "Toward an Understanding of the Motivation of Open Source Software Developers"

Eine anschauliche Erklärung des Aufbaus und der Funktionsweise von Open Source-Communities, versehen mit Handlungsempfehlungen.

6 Fazit: Open Source Software - ja oder nein?

Die vorangegangenen Kapitel haben den Leser von der Orientierungs-, über die Evaluations- und Implementierungs- hin zur Produktivphase durch den gesamten Prozess zur erfolgreichen Einführung von OSS geführt.

Innerhalb jedes Kapitels wurde neben einer Einführung in die vorhandene Theorie und hieraus abgeleiteten praxisnahen Empfehlungen am Beispiel der Migration einer Nachrichten- und Groupware aufgezeigt, wie sich die vorgeschlagenen Handlungsschritte konkret ausgestalten lassen.

Zu Beginn dieses Ratgebers wurde eine zusammenfassende Evaluationsmatrix vorgestellt, die bei einer abschließenden Entscheidung für oder gegen die Einführung einer konkret betrachteten OSS unterstützen soll. Diese Matrix wird hier noch einmal aufgegriffen und unter Zuhilfenahme der ausgefüllten Checklisten aus den einzelnen Kapiteln für unser Beispielunternehmen ausgewertet:

Entscheidungskriterium	Wert/Antwort	Relevanz	K.O.-Kriterium?
Ideologische Motivation (z.B. Freiheitsgedanke)	Für das Unternehmen zunächst nicht zutreffend.	0	NEIN
Einhaltung rechtlicher Rahmenbedingungen	Für das Unternehmen nicht zutreffend.	0	NEIN
Einschätzung Außenwirkung	Da die Nutzung der Groupware nicht nach außen kommuniziert wird, nicht relevant	0	NEIN
Einschätzung Innenwirkung	Da im Unternehmen viele nicht besonders technikaffine Mitarbeiter mit der Groupware arbeiten, besteht das Risiko, dass sich der Wechsel von der proprietären auf eine Open Source Lösung negativ auf das Betriebsklima auswirkt	4	NEIN
Kommunikation mit Partnern			

Ist die Software relevant für Schnittstellen mit Partnern?	Schnittstellen mit Partnern bestehen nur in der Form, dass E-Mails ausgetauscht werden.		
Wie hoch ist die Kompatibilität der Datenformate mit den Industriestandards?	Bei E-Mail-Formaten herrscht ein einheitlicher Standard, daher keine Ausirkungen		
Wie gut sind die Möglichkeiten, Datenformate zu ändern?	Hier nicht notwendig		
Gesamteinschätzung	Kommunikation mit den Partner nicht beeinträchtigt	10	JA
	Rechtliche Rahmenbedingungen		
Wird die Software modifiziert (oder weitervertrieben) und ergeben sich daraus rechtliche Konzequenzen?	Die Groupware soll nur intern unmodifiziert vervielfältigt werden. Daher keine rechtlichen Konsequenzen		
Stellt uns die Pflicht zur Freigabe von Quellcodeänderungen vor Probleme (Wahrung von Betriebsgeheimnissen)?	n.a.		
Gesamteinschätzung	Es entstehen aus der Nutzung der Groupware-Lösung keine rechtlichen Konsequenzen	6	NEIN
Gesamteinschätzung unmittelbare finanzielle Auswirkung	Einsparungen in Höhe von 41.190 €; positive finanzielle Auswirkungen in relevanter Größenordnung	10	JA
	Anforderungserfüllung		
Wie gut werden die funktionalen Anforderungen erfüllt?	Analog der Detailmatrix zu den funktionalen Anforderungen (Kapitel 3.1) werden diese bei der von uns gewählten Groupware erfüllt		
Wie gut werden die nicht-fuktionalen Anforderungen er-	Analog der Detailmatrix zu den nicht-funktionalen Anfor-		

füllt?	derungen (Kapitel 3.1) werden diese bei der von uns gewählten Groupware am besten erfüllt		
Gesamteinschätzung	Die ausgewählte Software erfüllt die Anforderungen in absolut hinreichendem Maße	10	JA
Qualität und Reifegrad von Software und Community			
Wie ist die Softwarequalität einzuschätzen?	Softwarequalität darf auf Grund der Referenzkunden als hoch eingestuft werden.		
Welchen Reifegrad hat die Community?	Projekt wird seit 2002 aktiv entwickelt. Die Community hat 2011 einen Community Award gewonnen, wird von einem Unternehmen unterstützt.		
Gibt es professionelle Supportangebote?	Es sind mehrere zertifizierte Anbieter vorhanden, die verschiedene Supportpakete anbieten.		
Gesamteinschätzung	Hohe Zuverlässigkeit von Software und Community ist gegeben.	8	JA
Migration			
Ist eine Migration technisch möglich?	Ja, eine Migrationsfabrik ist unterstützend vorhanden.		
Wird externer Sachverstand benötigt um die Migration durchzuführen?	Ja, ein Berater wird für die einwöchige Migration hinzugezogen.		
Gesamteinschätzung	Migration stellt keine Probleme dar.	6	JA

Tabelle 7: Entscheidungsmatrix ausgefüllt

Die Auswertung der Gesamtmatrix für das Referenzunternehmen des Ratgebers führt zu der Erkenntnis, dass durch die Einführung der Open Source-Groupware über einen Zeitraum von drei Jahren Kostenvorteile in Höhe von 41.190,00 € realisierbar sind. Diese Summe erscheint

(verglichen mit dem Jahresumsatz des Unternehmens) auf den ersten Blick nicht sehr hoch. Allerdings ließen sich mit beispielsweise Zertifizierungen für eine Vielzahl von Mitarbeitern bezahlen (Steigerung der Mitarbeiterzufriedenheit und des Knowhows im Unternehmen) oder sogar eine zusätzliche (Teilzeit-) Stelle schaffen (positive Außenwirkung). Zudem ist davonauszugehen, dass nach Ablauf der betrachteten drei Jahre eine Lizenzerneuerung der (alten) proprietären Software anstünde, die abermals Kosten verursacht. Nach Einführung der Open Source-Groupware würden diese entfallen. Auf lange Sicht ist also sogar von einer Steigerung der hier abgebildeten Einsparungen zu rechnen. Da die Gesamtmatrix darüber hinaus zeigt, dass keine Gründe gegen eine Einführung der OSS sprechen, wäre das Referenzunternehmen gut damit beraten, den Schritt zur Nutzung freier Software zu wagen.

Nach dem Lesen aller Kapitel, dem Nachvollziehen der Beispiele und der Anwendung der jeweils bereitgestellten Checklisten auf die eigene Organisation sollte der Leser in der Lage sein, diese Matrix (in angepasster Form) auch für seine konkrete Problemstellung auszufüllen und selbstständig zu entscheiden, ob die Einführung von OSS für das von ihm zu evaluierende Szenario Sinn ergibt.

Dies wird mit Sicherheit nicht für jede Organisation und jede Software der Fall sein. Im Verlauf der Kapitel wurde jeweils differenziert betrachtet, welche Argumente im Einzelfall für und welche gegen eine Einführung von OSS sprechen können. Es lässt sich jedoch festhalten, dass (ausgehend von der Zielgruppe des Ratgebers, die OSS zur Unterstützung ihres Tagesgeschäfts nutzen und maximal im Eigeninteresse modifizieren möchte) die Einführung von OSS in einer Vielzahl der Fälle äußerst erfolgsversprechend ist: Die Aussicht auf Kostenersparnisse, größeren (bzw. adäquateren) Funktionsumfang, freie Modifizierbarkeit und Wiederverwendbarkeit des Codes sowie eine breite Basis an frei verfügbarem Wissen als Motivatoren für die Einführung dürften bei ausreichender Planung nicht durch die resultierenden Risiken kompensiert werden. Nach dem Lesen dieses Ratgebers sollte der Leser außerdem mit dem nötigen Rüstzeug ausgestattet sein, etwaige Risiken einzuschätzen und ihnen adäquat zu begegnen. Möglicherweise ist damit für eine weitere Organisation der Weg geebnet, sich der Vorteile freier Software zu bedienen und den Gedanken offener Standards weiterzutragen.

7 Quellen

Bundesministerium des Inneren (2008): Migrationsleitfaden: Leitfaden für die Migration von Software, Version 3.0, 1 Auflage, Berlin

Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V. (o.J.): Open Source Software, Rechtliche Grundlagen und Hinweise, LEITFADEN Version 1.0, Berlin-Mitte

Computerschule (o.J.): URL: http://unispital-computerschule.uhbs.ch/informatikschule/schulung/index.jsp

Daffara, C. (2009): The Small/Medium Enterprise guide to Open Source Software, URL: http://guide.conecta.it/FLOSSguide.pdf

Esterházy, K., Schwab, W. (1998): TCO in der Diskussion bei führenden IT-Infrastruktur Lieferanten: Mehr als nur Produktfeatures?, in: Information Management, Nr. 2, S. 34 – 38

Geissdörfer, K., Gleich, R., Wald, A. (2009): Standardisierungspotentiale lebenszyklusbasierter Modelle des strategischen Kostenmanagements, in: Zeitschrift für Betriebswirtschaft, Nr. 6

Jaeger T., Metzger A. (2011): Open Source Software, Rechtliche Rahmenbedingungen der Freien Software, 3. Auflage, München: Beck Juristischer Verlag

Jungwirth, C./Luthiger, B. (2007): Pervasive Fun, URL: https://www1.ethz.ch/foss/people/lbenno/LuthigerJungwirth_PervasiveFun.pdf

Koglin, O./Metzger, A. (2004): Urheber- und Lizenzrecht im Bereich von Open-Source-Software

Krcmar, H. (2005): Informationsmanagment, 3. Auflagen, Berlin: Springer-Verlag GmbH

Krikpatrick, D., Krikpatrick L. (2008): Evaluating Training Programs, The Four Levels, 3. Auflage, San Francisco: Berrett-Koehler Publishers, Inc.

Luthiger, B. (2004): Alles aus Spaß?, Zur Motivation von Open-Source-Entwicklern, in: Open-Source-Jahrbuch 2004 (Hrsg. Robert A./Lutterbeck, B.), Berlin: Lehmanns Media, Seiten 93 – 106, URL: www.opensourcejahrbuch.de/Archiv/2004/pdfs/II-2-Luthiger.pdf

Mieritz, L., Kirwin, B. (2005): Defining Gartner Total Cost of Ownership

Orszag, P. (2009): Open Government Directive, MEMORANDUM FOR THE HEADS OF EXECUTIVE DEPARTMENTS AND AGENCIES, URL: http://www.whitehouse.gov/open/documents/open-government-directive

Reasoning Inc. (o.J.): How Open Source and Commercial Software Compare: MySQL 4.0.16, A Quantitative Analysis of Database Implementations in Commercial Software and in MySQL 4.0.16, URL: http://www.reasoning.com/pdf/MySQL_White_Paper.pdf und A Quantitative Analysis of TCP/IP Implementations in Commercial Software and the Linux Kernel, URL: http://www.reasoning.com/downloads/opensource.html

Riepl, L. (1998): TCO versus ROI, in: Information Management, 13. Jg., Nr. 2, S. 7-12

Roberts, J. et al. (2006): Understanding the Motivations, Participation and Performance of Open Source Software Developers: A Longitudinal Study oft he Apache Projects, in: Management Science, 52. Jg., Nr. 7

Shah, S. (2006): Motivation, Governance and the Viability of Hybrid Forms in Open Source Software Development, in: Management Science, 52 Jg., Nr. 7, Seite 1000 – 1014, URL: http://faculty.washington.edu/skshah/Shah%20-%20Motivation,%20 Governance,%20Hybrid%20Forms.pdf

Sladic, D. (2005): Durchführung eines Anwendertests in der Praxis, Pragmatische Ansätze und Tools zur Testplanung, Teststeuerung und Fehlerkommunikation, in: 20. WiMAW-Rundbrief, 11 Jg., Nr. 2, URL: http://www.ifs-consulting.com/_downloads/ WIMAW_PUB_Anwendertest_20050729_IFSC.PDF, Abruf: 30.12.2011

Sneed, H., Wolf, E., Heilmann, H. (2010): Softwaremigration in der Praxis, Übertragung alter Softwaresysteme in eine moderne Umgebung, 1. Auflage, Heidelberg: dpunkt.verlag GmbH

Wild, M., Herges, S. (2000): Total Cost of Ownership (TCO) – Ein Überblick, URL: http:// geb.uni-giessen.de/geb/volltexte/2004/1577/pdf/Apap_WI_2000_01.pdf

Ye, Y., Kishida, K. (2003): Toward an Understanding of the Motivation of Open Source Software Developers, Proceedings of the 25th International Conference on Software Engineering, URL: http://l3d.cs.colorado.edu/~yunwen/papers/ICSE03.pdf